VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Využití spektrálního shlukování pro segmentaci obrazu

The Use of Spectral Clustering for Image Segmentation

2012 Bc. Tomáš Bajar

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Tomáš Bajar**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Využití spektrálního shlukování pro segmentaci obrazu
The Use of Spectral Clustering for Image Segmentation

Zásady pro vypracování:

Cílem diplomové práce je ověřit možnosti segmentace obrazu s využitím spektrální dekompozice Laplaceovy matice sítě, kterou obraz vytváří, s následným shlukováním. V diplomové práci proveďte:

1. Seznamte se s problematikou spektrálního shlukování (vhodným materiálem je např.: Ulrike von Luxburg, A Tutorial on Spectral Clustering).
2. Metodu spektrálního shlukování implementujte a experimentálně ověřte pro segmentaci obrazu. Pozornost věnujte efektivnímu postupu výpočtu vlastních čísel a vektorů Laplaceovy matice (lze použít např. knihovnu ARPACK), jakož i výběru vhodné shlukovací metody.
3. Chování metody otestujte a shrňte dosažené výsledky.

Programátorské práce proveďte v C/C++.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Dr. Ing. Eduard Sojka**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Rád bych na tomto místě poděkoval svému vedoucímu diplomové práci doc. Dr. Ing. Eduardu Sojkovi za jeho čas, vedení mé práce a rady, které mi vždy trpělivě a ochotně poskytl.

V Ostravě 30. Června 2012 .....

Abstrakt

Spektrální shlukování je metoda, která dokáže na bázi zpracování vlastních čísel a jejich vlastních vektorů matice podobnosti pro vstupní body rozdělit tyto vstupní body na jednotlivé oblasti. V minulosti byla tato metoda úspěšně aplikována na segmentaci různých prvků mimo obor zpracování obrazu. Tato diplomová práce si klade za cíl prozkoumat tuto metodu a experimentálně ověřit její použitelnost pro potřeby segmentace digitálního obrazu. Práce se také zaměří na popis vlivu nastavení vstupních parametrů této metody na výsledek segmentace, jelikož tento vliv není v dostupné literatuře příliš podrobně popsán. V rámci této práce bude také ukázáno, jaké knihovny lze použít pro efektivní výpočet vlastních čísel a jejich vlastních vektorů. Pozornost bude také věnována rozboru implementace programu využívajícího metodu spektrálního shlukování a detailnímu popisu jeho dosažených výsledků.

Klíčová slova: spektrální shlukování, vlastní čísla, vlastní vektory, segmentace, ARPACK, OpenCV

Abstract

Spectral clustering is a method which can separate input points into individual regions on the basis of eigenvalues and their corresponding eigenvectors of similarity matrix made for these input points. This method has been successfully applied on the segmentation of various input points outside the domain of image analysis. This diploma thesis aims to explore this method and experimentally verify its usability for the needs of digital image segmentation. The thesis also aims to describe the influence of setting the input parameters for this method on the results, because this influence is not described in too much detail in the available literature. In the framework of this thesis it will also be shown, which libraries could be used for effective computation of eigenvalues and their eigenvectors. Attention will also be paid to the analysis of implementation of such program using the method of spectral clustering as well as its results.

Key words: spectral clustering, eigenvalues, eigenvectors, segmentation, ARPACK, OpenCV

Použité zkratky a pojmy

EM Clustering	- Expectation Maximization Clustering, shlukovací metoda
ARPACK	- Arnoldi Package, základní verze knihoven
ARPACK++	- Arnoldi Package C++, verze knihoven s rozhraním objektů jazyka C++
PARPACK	- Parallel Arnoldi Package, verze knihoven s podporou paralelizace výpočtů
OpenCV	- Open Computer Vision, knihovny pro práci s obrazem
SuperLU	- Super Lower Upper, knihovny pro trojúhelníkový rozklad matic
UMFPACK	- Unsymmetric multifrontal sparse LU factorization package
FORTRAN77	- Programovací jazyk Fortran, verze 77
DLL	- Dynamically Linked Libraries, dynamické knihovny systému Windows
MB	- Mega Byte, 10^6 bajtů
GB	- Giga Byte, 10^9 bajtů
RGB	- Red Green Blue, posloupnost barev subpixelů
BGR	- Blue Green Red, zpětná posloupnost barev pixelů
CSC	- Compressed
GHz	- Giga Hertz, 10^9 hertzů
CPU	- Central Processing Unit, procesor počítače

Seznam obrázků, ukázek kódu, vztahů a grafů

Obsah

1 Úvod

Tato diplomová práce se zabývá segmentací obrazu. Pro člověka se jedná o zcela intuitivní věc, kterou zvládá v každodenním životě bez větších obtíží. Pro počítač, či jiné přístroje, je však segmentace obrazu něco, co ve svém základu neumí. Pro počítač jsou vstupní data jakéhokoli formátu obecně pouhými čísly, která pro něj postrádají význam a ze kterých si sám neumí odvodit informace pro své porozumění. Je tedy na člověku, aby počítači předal instrukce a přesný postup, jak tohoto porozumění vstupním datům dosáhnout. Tato diplomová práce se tedy věnuje určité metodě, která počítači umožní najít v obrazu informace.

Pod pojmem segmentace obrazu rozumíme metodu, která ve vstupním obrazu hledá jeho části, které dávají smysl v námi požadovaném kontextu. Jedná se tedy především o identifikaci objektů, které tvoří popředí. Tyto objekty jsou středem našeho zájmu nejčastěji z důvodu dalšího zpracování pro získání dalších konkrétních informací. Tyto objekty hledáme převážně na základě jistých společných vlastností, které jsou pro ně typické.

I pro člověka, který toto velmi dobře intuitivně ovládá, však nejsou tyto metody mnohdy jednoduché. Segmentace obrazu se týká mnoho metod, které mají rozličné základy. Ty nejjednodušší z nich nedělají člověku žádný problém, co se týče jejich zachycení v podobě programového kódu. Problémem těchto jednoduchých metod jsou však jejich nevhodné výsledky při použití na reálně řešených problémech z praxe. Vstupní obraz samozřejmě mnohdy není ideální a obsahuje mnoho faktorů, které segmentaci značně znesnadňují. Dále také i objekty popředí, které se snažíme ve vstupním obrazu najít, obsahují mnohdy více částí, přechodů, či nejasných hran. Vzhledem ke všem těmto rušivým faktorům bývají pokročilé segmentační metody ve svém základě vcelku komplexní, aby byly schopny si poradit s co možná největším počtem těchto negativních vlivů. Tyto netriviální metody však produkují obvykle lepší výsledky, což je hlavním důvodem jejich zkoumání a zlepšování.

Tato diplomová práce se bude věnovat jedné z takových metod, která se nazývá spektrální shlukování. V následujících kapitolách bude vysvětleno, na jakém principu tato metoda funguje. Dále se podíváme na to, jakým způsobem se mi podařilo tuto metodu implementovat a jaké jsou její výsledky. Důraz bude věnován také vlivu vstupních parametrů této metody na výsledky, což je oblast, která není zcela dobře probádána a bude tudíž určitě velmi přínosná.

2 Metody spektrálního shlukování

Této segmentační metody se týká více postupů, které je možno aplikovat do praxe. Základ metody bude podrobně vysvětlen v další kapitole s ohledem na to, jak jsem jej aplikoval já ve své práci. Existuje však řada alternativních postupů, které byly v minulosti prozkoumány a zdokumentovány. Tato kapitola bude tedy probírat dostupné alternativy této metody.

2.1 Optimální řez grafu

Tato nejznámější metoda, která byla v praxi již mnohokrát zdokumentována, se soustředí na práci s druhým nejmenším vlastním číslem matice podobnosti a jeho vlastním vektorem. Její základy pocházejí z prvních prací o dělení grafu pomocí spektrální analýzy jeho matice podobnosti kolem roku 1973 [3]. Pomocí hodnot obsažených v tomto jednom vlastním vektoru je proveden optimální řez vstupního grafu, který je reprezentován právě maticí podobnosti. Tím vstup optimálně rozdělíme na dvě oblasti. Jak úspěšně jsme dosáhli optimálního rozdělení, lze zjistit pomocí výpočtu řezového koeficientu, který popisuje velikost nalezených oblastí [10]. Pokud bychom chtěli dostat více oblastí, aplikujeme dále tento postup rekurzivně na získaný výsledek.

2.2 Použití více vlastních vektorů

Další metody se věnují použití více vypočtených vlastních čísel a jejich vlastních vektorů [7] pro účely segmentace vstupu. Tyto metody byly zkoumány a popsány například v kontextu segmentace zákazníků podle jejich nákupů, referencí v dokumentu [8] a podobně. Také u těchto metod existují stanovené metriky, kterými je dobré se při rozdělování grafu řídit. Dobře popsány jsou tyto vztahy a operátory pro minimalizaci například v dostupné literatuře jako [7] nebo [11].

Záleží na počtu vlastních vektorů, které jsou následně zpracovány shlukovacím algoritmem. Některé algoritmy používají menší počet vlastních vektorů, které volí vzhledem k nejnižším hodnotám jejich vlastních čísel, jak bylo popsáno například v [4] nebo [5]. Přesně této verzi se věnuje má práce. Počet je závislý na předchozí volbě, ke které se v literatuře nevyskytuje příliš mnoho informací. V mé práci jsem se tomuto experimentálně věnoval. Jiné algoritmy zase volí vlastní vektory náležící největším hodnotám vlastních čísel [1]. Existují i metody, u kterých se mluví o použití co možná největšího počtu vypočtených vlastních čísel a jejich vektorů [5].

2.3 Podobnost prvků

Důležitou roli pro metodu spektrálního shlukování hraje stanovení podobnosti mezi vstupními body. Podmínkou je co nejlepší rozdělení hodnot podobnosti mezi body tak, aby hodnota byla vysoká pro podobné body a co nejmenší pro body, které si podobny nejsou [7]. Obecně se nejčastěji používá funkce normálního rozložení, kterou známe ze statistiky. Místo ní bychom však mohli použít i jinou funkci, která by podobnost dobře modelovala, což ale záleží na konkrétní aplikaci.

Je navíc nutno stanovit, na základě čeho budeme body porovnávat. Můžeme totiž porovnávat všechny body mezi sebou, nebo jen jejich část. Můžeme porovnávat body v určitém předem stanoveném okolí, či stanovit maximální počet nejbližších bodů. To, jak a které body budeme porovnávat, je třeba vybrat na základě jistého modelu, který charakterizuje naši úlohu. O metodách výběru bodů pro porovnávání

se dále rozepisuji v kapitole o matici podobnosti. Po zvolení modelu, který vystihuje naši úlohu, tedy můžeme sestrojit pomocí podobnostní funkce matici podobnosti. Zde máme poté dále na výběr, jakou cestou z ní vytvoříme Laplaceovu matici. Volba tvaru Laplaceovy matice může rovněž být volena s ohledem na danou aplikaci, aby lépe vystihovala podstatu, ze které vychází. V literatuře jsou zmíněny dva nejvíce používané typy Laplaceovy matice [7]. Jedná se o normalizovaný a nenormalizovaný tvar. Pro účely své práce jsem volil tvar nenormalizovaný, neboť přesněji vystihuje fyzikální podstatu mé úlohy, která bude popsána v dalších kapitolách. Nenormalizovaný tvar navíc, na rozdíl od normalizovaného, produkuje symetrickou Laplaceovu matici, čehož lze v následném zpracování využít a v matici se snáze orientuje. U normalizovaného tvaru se navíc nejčastěji setkáme s postupem, který zahrnuje výpočet inverzní matice, což je pro velké vstupní matice výpočetně náročný úkol. Použití normalizované Laplaceovy matice má jisté drobné výhody, avšak výrazné zlepšení výsledků není dokumentováno.

3 Můj přístup ke spektrálnímu shlukování

Metoda segmentace, kterou se tato diplomová práce zaobírá, se jmenuje spektrální shlukování s *k*-cestným rozdělením. Tento název vyplývá z podstaty této metody, která má kořeny v matematice. Konkrétně tato metoda využívá vlastnosti spektra matice, se kterým dále pracuje. Spektrem matice rozumíme v tomto případě vlastní čísla dané matice společně s vlastními vektory, které odpovídají jednotlivým vlastním číslům. Metoda spektrálního shlukování se snaží rozdělit vstupní body mezi *k* segmentů tak, aby body uvnitř každého segmentu měly mezi sebou velkou podobnost a aby podobnost mezi body z jiných segmentů byla malá. Toto rozdělení bodů je založeno právě na práci s vlastními čísly a vlastními vektory podobnostní matice. Podobnostní matici pro vstupní obraz se věnuje následující podkapitola.

Princip metody spočívá ve správném výběru vlastních čísel a korespondujících vlastních vektorů pro další zpracování. Tento správný výběr není předem známý, jak bude viditelné z experimentální části mé práce. Počet vybraných vlastních čísel a jejich vlastních vektorů totiž zásadně ovlivňuje konečný výsledek segmentace. Ohledně jejich výběru existují jistá doporučení a postupy, které vycházejí z předchozí praxe vědců věnujících se této problematice. Některá hlavní doporučení budou v této mé diplomové práci rozebrána a ukázána na konkrétních příkladech. Především se bude jednat o význam druhého nejmenšího vlastního čísla a jemu odpovídajícímu vlastnímu vektoru, na bázi čehož je přímo založena jedna z metod spektrálního shlukování. Autoři, kteří tento postup vyzkoušeli, však došli k závěru, že za určitých podmínek je lepší volit jiný přístup. Tím je zaměřit se na zpracování více vlastních čísel a vlastních vektorů jím odpovídajících najednou pro účely shlukování. Právě tomuto přístupu ke spektrálnímu shlukování se má práce věnuje nejvíce.

Po vybrání určitého počtu vlastních čísel společně s jejich vlastními vektory obvykle následuje jejich přepočet, který bude podrobněji popsán v jedné z dalších podkapitol. Po přepočtení budou tyto vektory dané segmentační metodě sloužit pro snadnější určení segmentů, ke kterým patří původní body vstupního obrazu. Algoritmus, kterým se tyto nové souřadnice tvořené vypočtenými vlastními vektory budou shlukovat, lze také volit v závislosti na aplikaci. Právě tento shlukovací algoritmus, použitý na konci postupu metody spektrálního shlukovaní, má také nezanedbatelný vliv na kvalitu výsledku. Existují mnohé známé shlukovací algoritmy, které byly úspěšně nasazeny v praxi a jejichž popis včetně ukázek programového kódu lze nalézt. Většinou jsou však takové ukázky či hotové postupy pevně svázány s prací v předem stanovené dimenzi. Díky tomu jsem je nemohl využít ve své práci, neboť shlukovací algoritmus, který je pro spektrální shlukování potřebný, musí umět pracovat obecně v jakémkoli počtu dimenzí. Počet dimenzí pro koncové shlukování je totiž dán právě zvoleným počtem vlastních čísel a jejich vlastních vektorů. O shlukovacích algoritmech pro tento problém je tedy psáno v poslední podkapitole této sekce.

3.1 Matice podobnosti

Práce segmentační metody spektrálního shlukování je založena na práci s maticí podobnosti. Tato matice je tvořena hodnotami, které představují míru toho, jak jsou si dva body dané vstupní množiny bodů mezi sebou podobny. Spektrální shlukování je metoda, která má spíše obecný původ, který nepochází z oblasti zpracování obrazu. Je tak pomocí ní možno segmentovat i různé jiné množiny

bodů, než pixely obrazu. Právě jednotlivými pixely vstupního obrazu pro segmentaci se budu ve své práci zabývat. Obecně však tato metoda může pracovat i s jinou reprezentací podobnosti vstupních prvků jako například grafy různých typů. Ve své podstatě budu pro své účely graf tvořit i já, avšak jeho reprezentaci budu realizovat právě maticí podobnosti. V té budou na odpovídajících pozicích uloženy hodnoty pro jednotlivé hrany mezi vrcholy grafu. Do polí podobnostní matice tedy budeme ukládat hodnotu, která bude určovat podobnost dvou pixelů. Volba typu grafu, který budeme ze vstupní množiny bodů konstruovat, může také mít vliv na výsledek algoritmu, ačkoli konkrétní poznatky v této oblasti chybí.

Obecně se pro metodu spektrálního shlukování používají tyto typy grafů:

1. Graf ε-okolí

V tomto grafu propojujeme všechny body vstupní množiny do vzdálenosti ε od aktuálně zpracovávaného bodu. Jelikož mají body tu vlastnost, že náleží malému okolí o maximální vzdálenosti ε, počítání jejich vzájemných podobností by nám mnoho dalších informací nepřineslo. Tento graf je tedy obvykle konstruován jako nevážený, to jest bez hodnot náležících hranám.

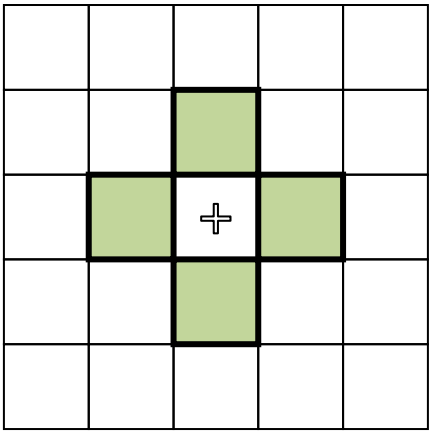
2. Graf k nejbližších sousedů

V tomto grafu se snažíme propojit zpracovávaný bod s jeho k nejbližšími sousedy. Jelikož toto sousedství nemusí platit pro nalezené body z druhé strany, vzniká tímto postupem orientovaný vážený graf. Pokud se tomuto chceme vyhnout, můžeme si vybrat ze dvou postupů. Prvním je tuto asymetrii zanedbat a hranu mezi body vždy udělat. Druhým je zapsání hrany pouze v případě, že sousední nalezený pixel rovněž obsahuje právě zpracovávaný bod mezi svými sousedy.

3. Plně propojený graf

U tohoto grafu jednoduše propojíme hranou všechny body vstupní množiny, pokud mají nezápornou hodnotu podobnosti. Takový graf by samozřejmě měl obsahovat informace o vztazích jednotlivých bodů vzhledem k malým oblastem, které tvoří. Je proto vyžadováno, aby váhová funkce, kterou počítáme podobnost mezi pixely, modelovala tyto malé lokální oblasti. Tuto vlastnost má například známá funkce Gaussiánu.

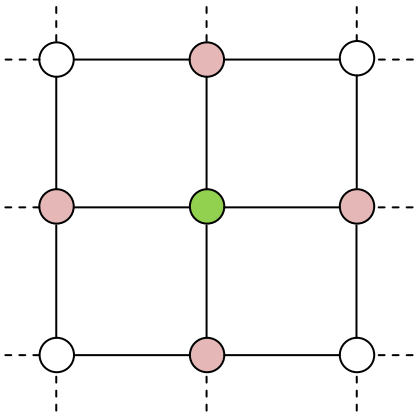
Pro potřeby této práce jsem zvolil způsob, který vychází z prvních dvou uvedených. Jelikož se pohybujeme v obraze tvořeném jednotlivými pixely, počítáme podobnost mezi nejbližšími čtyřmi sousedy, jak znázorňuje obrázek 1.



1 – Princip čtyřsousednosti pro nabírání okolních pixelů

3.2 Laplaceova matice

Další typickou hlavní věcí pro běh metody spektrálního shlukování je Laplaceova matice. Tuto matici sestrojíme z předešlé matice podobnosti. Je dobré zmínit, že dle literatury, která o těchto Laplaceových maticích pojednává v širší míře, jaksi neexistuje jednotný ustálený způsob, jak konkrétně takovou matici sestrojit. Různí autoři, kteří se ve svých pracích snaží pojednat o využití Laplaceových matic například pro účely spektrální segmentace, mají často pod tímto pojmem svou vlastní verzi této matice s mírnými odlišnostmi oproti ostatním. Zatímco ústřední myšlenka je opřená o pevné základy, její praktické provedení pro využití v analýze obrazu se může v detailech poměrně lišit dle autora či dle požadované funkčnosti. V podstatě se však jedná o přepis Kirchhoffových zákonů pro uzel sítě, kde zapisujeme proudy smyček. V mém případě se pro potřeby této práce pracuje s uzlem a okolními hodnotami dle přiloženého obrázku 2.



2 – Aplikace fyzikálního modelu po potřeby sestrojení Laplaceovy matice

Počítáme-li hodnoty pro aktuálně zpracovávaný uzel, který je na obrázku znázorněn zelenou barvou, dosazujeme na příslušné pozice do Laplaceovy matice hodnoty odpovídající proudům tekoucím ze sousedních uzlů, které jsou na obrázku znázorněny barvou červenou. Podle směru proudu, který může téci buď z uzlu ven, nebo do něj, pak volíme pro příslušnou hodnotu znaménko. Hodnoty, které doplňujeme do Laplaceovy matice, se řídí následujícím předpisem:

$$L_{i,j} = \begin{cases} deg(v_i) & pro\ i = j \\ -1 & pro\ i \neq j \wedge i\ je\ sousedem\ j \\ 0 & pro\ ostatní \end{cases}$$

3 – Předpis pro hodnoty polí Laplaceovy matice

Hodnotu -1 bychom ale v našem případě zapisovali pouze za podmínky, že by sousední pixely měly zcela stejnou barvu. Jelikož tomu tak v reálných vstupních obrazech bude málokdy, budeme místo této hodnoty zapisovat do Laplaceovy matice číslo určující podobnost těchto dvou sousedních pixelů. Tuto hodnotu budeme vypočítávat pomocí podobnostní funkce realizující normální rozdělení, jehož grafem je známý Gaussián. Taková funkce nám bude zároveň dobře modelovat ono blízké okolí zmíněné v předchozí kapitole. Do polí Laplaceovy matice tedy budou uloženy hodnoty podobnosti pixelů, které se vypočtou dle následujícího předpisu. V čitateli exponentu nalezneme umocněný rozdíl barev právě počítaných pixelů v_i a v_j. Místo něj bychom mohli dosadit váhu z předešlé vypočtené matice podobnosti. Jelikož však ve své práci a především pak programovém kódu počítám tuto hodnotu rovnou, uvedený vzorec je tak výstižnější.

$$P_{i,j} = e^{-\frac{(v_i - v_j)^2}{2\sigma^2}}$$

4 – Vzorec pro výpočet hodnoty podobnosti pixelů

Tato hodnota se tak bude dle předpisu nacházet na pozicích Laplaceovy matice tam, kde bude odpovídat sousedícím pixelům. Na hlavní diagonále se pak budou dle předpisu vyskytovat hodnoty $deg(v_i)$, které se vypočítají sečtením všech hodnot na daném řádku a změnou znaménka.

Laplaceova matice má pro segmentační metodu spektrálního shlukování hlavní význam díky svým vlastnostem, které pro účely následné segmentace přidává. Jedná se například o vlastnost, která říká, že taková matice je pozitivně semidefinitní, tudíž všechna vlastní čísla z ní vypočtená budou větší, nebo rovna nule. Tato vlastnost mi například sloužila ke kontrole při programování výpočtu těchto vlastních čísel. Dále se jedná o zajímavou vlastnost, která říká, že kolikrát se nulová hodnota vlastního čísla objeví ve výsledcích Laplaceovy matice, tolik propojených komponent původní graf obsahuje. Tato vlastnost je také využitelná pro potřeby segmentace.

3.3 Vlastní čísla a vlastní vektory

Po vytvoření Laplaceovy matice je dalším krokem postupu vypočtení jejich vlastních čísel společně s odpovídajícími vlastními vektory. Jedná se o hlavní krok metody spektrálního shlukování. Výsledek tohoto kroku přímo ovlivňuje segmentační výsledek, neboť záleží jaká vlastní čísla a vlastní vektory vezmeme v potaz pro další zpracování. Dle metod zmíněných v jedné z předchozích kapitol lze pro následné finální shlukování vybrat jeden či více vlastních vektorů. Právě výběrem těchto vlastních vektorů a jejich počtem se budu ve své práci zabývat a popisovat vliv, který mají na výsledek. Vlastní čísla a vektory vstupní Laplaceovy matice vychází z následujícího předpisu.

$$Lx - lx = 0$$

5 – Rovnice pro výpočet vlastních čísel s vlastními vektory

Kde L označuje vstupní Laplaceovu matici, x jeden z vlastních vektorů a l vlastní číslo korespondující s daným vlastním vektorem.

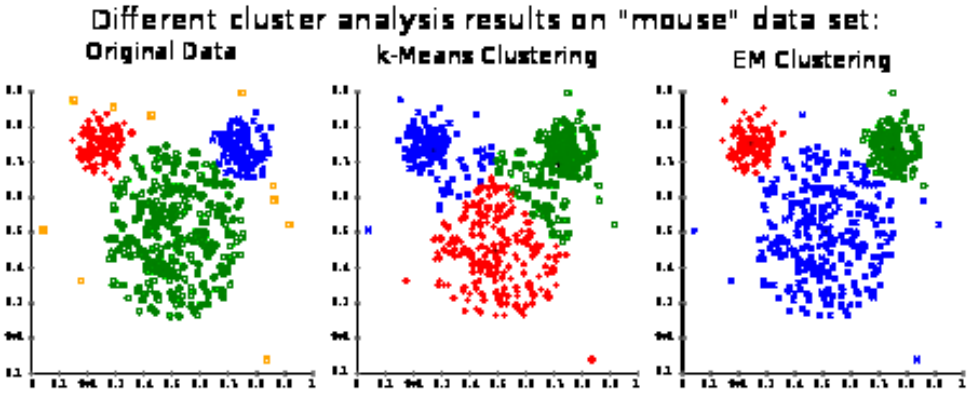
3.4 Nové souřadnice a jejich transformace

Z předešlého kroku výpočtu jsme tedy obdrželi vlastní čísla společně s jejich vlastními vektory. Poté co zvolíme určitou množinu vlastních čísel s odpovídajícími vlastními vektory, je obvykle prováděna jejich transformace. Tou je myšleno to, že vybrané vlastní vektory, které jsme získali zapsané v řádcích, zapíšeme místo řádků do sloupců. Z pohledu matice se tedy jedná o jejich transpozici. Takto nově uspořádané vlastní vektory nám nyní budou sloužit jako nové souřadnice pro body vstupního obrazu. Počet zvolených vlastních vektorů tedy tímto způsobem určuje dimenzi, ve které se budou nové koordináty nacházet. Právě z tohoto důvodu potřebujeme pro následující finální shlukování metodu, která umí pracovat obecně pro jakýkoli počet dimenzí.

Dále se pak s těmito řádky nových souřadnic provádí operace normalizace, která všechny z nich přepočte na jednotkovou délku. Tuto operaci provádíme v závislosti na tom, jaký zvolíme postup a také jakou Laplaceovu matici předložíme na vstup metody výpočtu vlastních čísel a vektorů. Různé výpočetní metody totiž dávají rozdílné výsledky vlastních čísel a vektorů v závislosti na tom, jaký software pro jejich výpočet použijeme.

3.5 Shlukování

Nové souřadnice ke vstupním bodům obrazu, které jsme obdrželi z minulého kroku, obsahují oproti původní reprezentaci pomocí matice rozšířené vlastnosti, jež se týkají určování segmentů, ke kterým patří. Tyto nové vlastnosti plynou z použití Laplaceovy matice a výpočtu jejich vlastních čísel a vektorů. Jako poslední krok této segmentační metody provedeme klasické shlukování pomocí již známých metod. Volba shlukovací metody pro finální určení segmentů rovněž ovlivňuje výsledek. V mém případě jsem se v této práci zaměřil pouze na jednu z nich. Zvolil jsem známou metodu k-means, která byla doporučena i v literatuře, ze které jsem čerpal informace. Tato metoda bude detailněji popsána v dalších kapitolách včetně způsobu, jakým jsem ji implementoval. Pro obdržení co nejlepších výsledků se však doporučuje použít některou z komplexnějších metod. Následující obrázek 6 ukazuje pro představu, jakým způsobem se mohou výsledky finálního shlukování zlepšit při použití alternativních metod.



6 - Ukázka rozdílu výsledků finálních shlukovacích metod

Na obrázku je vidět snaha mnou použité shlukovací funkce k-means, která se snaží zachovat v hustě obsazené množině prvků stejnou velikost mezi nalezenými segmenty. Komplexnější metoda, označovaná v obrázku jako EM Clustering, je založena na výpočtu segmentů pomocí normálního rozložení. Díky tomuto faktu vrací tato metoda lepší výsledky, jelikož lépe zohledňuje původní rozložení vstupních vzorků, které bylo tímto způsobem generováno. Pro mé potřeby však první metoda shlukování k-means stačila.

4 Použité technologie

V této kapitole budou popsány technologie, které jsem ve své práci použil. Jedná se především o veřejně dostupné programové knihovny pro jazyk C/C++. Jelikož jsou součástí metody spektrální segmentace kroky, které vyžadují netriviální algoritmy, rozhodl jsem se sáhnout po již existujících řešeních namísto vlastní implementace. Hlavním důvodem pro toto rozhodnutí byla jistě jak časová úspora, tak i kvalita výsledků. Pro práci s obrazem jsem zvolil knihovny OpenCV, které znám ze svého předchozího studia předmětů věnujících se této problematice. Pro řešení problematiky vlastních čísel a vlastních vektorů jsem po konzultaci s vedoucím mé práce zvolil knihovny ARPACK. V následujících podkapitolách budou tyto knihovny popsány podrobněji.

4.1 OpenCV

Zkratka OpenCV znamená ve svém celém znění Open Computer Vision. Jedná se o programové knihovny, které se obecně zaměřují na práci s obrazem a jejich cílem je usnadnit a sjednotit přístup k řešení úloh v této oblasti. Tyto knihovny byly vyvíjeny od roku 1999 společností Intel pro podporu řešení grafických úloh jako například sledování paprsku v reálném čase a podobně. Od roku 2008 převzala jejich vývoj společnost Willow Garage, která se podílí na vylepšování těchto knihoven dodnes. Tyto knihovny nacházejí v praxi široké využití a můžeme se s nimi setkat v algoritmech, jako je rozpoznání obličejů, prostorové transformace a podobně. Použití těchto knihoven je možno vidět i ve spojitosti s jinými známými technologiemi jako jsou pohybový senzor Microsoft Kinect, či různá odvětví robotiky, což určitým způsobem dokladuje jejich kvalitu.

Do mého projektu jsem tyto knihovny, které jsou psány v jazyce C, přidal pomocí balíčkovacího systému mého operačního systému Ubuntu stejně jako další knihovnu, která je popsána v podkapitole níže.

Pro účely mého programu jsem z těchto knihoven využil především:

- Snadné nahrávání obrázků z pevného disku
- Rychlý přístup k jednotlivým pixelům obrázku
- Okna pro zobrazení grafického výstupu z programu
- Algoritmy pro změnu vnitřní reprezentace hodnot obrázku
- Základní operace s obrázky jako například zmenšování a zvětšování

4.2 ARPACK

Název této knihovny je zkráceninou názvu Arnoldi Package. Jedná se o knihovny, které slouží pro obecný výpočet vlastních čísel a jim odpovídajícím vlastních vektorů pro vstupní matici. Tyto knihovny jsou navrženy primárně pro práci s rozsáhlými strukturovanými maticemi s řídkým obsazením hodnot. Pod pojmem strukturovaná matice rozumíme takovou matici, ve které se hodnoty vyskytují na určitých typech míst. Může se například jednat o matici pásovou, kde nenulové hodnoty představují pásy rovnoběžné s hlavní diagonálou matice a podobně. Matice s řídkým obsazením hodnot znamená takovou matici, kde nenulové prvky tvoří pouze malý zlomek obsahu. Laplaceova matice, jejíž konstrukce je popsána výše, je přesně tímto typem matice. Obsahuje procentuálně velmi

málo nenulových prvků. Všechny tyto nenulové prvky pak v matici tvoří přesně rovnoběžné pásy s hlavní diagonálou. Vzhledem k těmto faktům byl výběr knihoven ARPACK jistě vhodnou volbou.

Tyto knihovny nabízejí celou řadu metod pro řešení mnoha problémů týkajících se výpočtu vlastních čísel a vektorů. Společně jsou však založeny především na algoritmické výpočetní metodě Arnoldiho procesů, konkrétněji Implicitně Restartované Arnoldiho Metody (IRAM), a v případě symetricity vstupní matice pak Implicitně Restartované Lanczosovy Metody (IRLM). Pro komplexnější problémy je potom využívána metoda faktorizace vstupní matice.

Havními rysy ARPACKu jsou:

- Zpětné komunikační rozhraní (bude popsáno podrobněji v kapitolách o implementaci)
- Manuální nastavení počtu požadovaných vlastních čísel a vektorů
- Možnost volby, zda požadujeme nejmenší či největší vlastní čísla a jejich vlastní vektory
- Možnost standardní (float) či dvojité (double) přesnosti výpočtu
- Možnost řešit standardní či zobecněné problémy
- Možnost volby strategie pro řešení daného problému
- Připravené struktury pro zadávání vstupních matic (bude popsáno podrobněji v kapitolách o implementaci)

Knihovny ARPACKu se pak opírají o funkci těchto dvou podknihoven:

1. SuperLU – Tato volně dostupná knihovna je zaměřena na přímé řešení velkých řídkých nesymetrických soustav lineárních rovnic. Je psána v jazyce C a její metody lze volat jak z jazyka C, tak i Fortranu. Funkce je založena na metodě dekompozice matice na horní a dolní trojúhelníkové oblasti, které jsou posléze řešeny dopřednou a zpětnou substitucí.
2. UMFPACK – Jedná se také o knihovny poskytující metody pro výpočet řídkých nesymetrických soustav lineárních rovnic, avšak s doplněním funkcí pro faktorizaci matic. Jsou napsány v jazyce C a jsou také volně dostupné.

Samotné knihovny balíku ARPACK jsou psány v programovacím jazyce Fortran77, jelikož jejich vývoj začal před mnoha lety. Aby bylo možné tyto efektivní a propracované algoritmy bez obtíží využívat v novějších programovacích jazycích, existuje řada nadstaveb nad tímto základním balíkem. Jelikož jsem celý program pro segmentaci pomocí metody spektrálního shlukování psal v jazyce C/C++, zvolil jsem pro své účely balík ARPACK++. Ten nabízí přístup k původním metodám psaným ve Fortranu skrze objektová rozhraní jazyka C++. Dále je pak možno pracovat s verzí balíku PARPACK, který přináší výhodu paralelizace běhu svých procedur a tím pádem lepší časy výpočtu. Tento paralelní balík jsem pro své potřeby nevyužil.

Tyto knihovny, ať už v jakémkoli balíku, je samozřejmě možno využívat na jakémkoli operačním systému, avšak zde považuji za dobré zmínit jistá úskalí, která jsem objevil při jejich instalaci a zprovozňování ve svém projektu. Nejjednodušší cestou je nainstalovat si celý předem připravený balíček (například ARPACK++) skrze balíčkovací systém Unixového operačního systému. Tím se knihovny rozbalí na správná místa do předem připravené hierarchie složek, odkud je potom možné je

jednoduše linkovat do projektu skrze zvolené vývojové prostředí. Pro zprovoznění těchto balíků na systémech Windows je potřeba nejprve stáhnout určité programy pro jejich převedení do použitelné podoby. Je potřeba vlastnoručně překompilovat zdrojové kódy a poté je v dalším nástroji převést do podoby DLL knihoven. Celý tento postup je dle mého značně neintuitivní a zbytečně složitý. Z tohoto důvodu jsem volil snazší alternativu skrze balíčkovací systém operačního systému Ubuntu, na kterém jsem knihovny ARPACKu zprovoznil v rozumném čase.

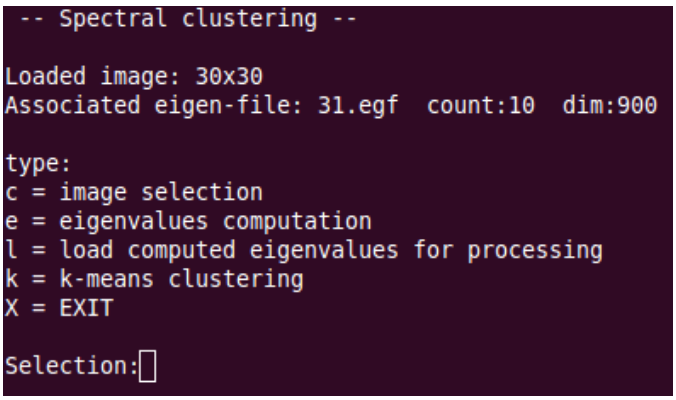
5 Algoritmus

Mým úkolem bylo dle zadání ověřit segmentační možnosti metody spektrální dekompozice s následným shlukováním. Z dostupné literatury o tomto tématu lze snadno zjistit, že tato metoda podává dobré výsledky na poli rozdělování a segmentace grafů. Mým cílem bylo tedy vyvinout takový algoritmus, který by tyto výsledky aplikoval do oblasti práce s digitálním obrazem. I tato problematika byla samozřejmě již určitým způsobem zpracována v minulosti a její výsledky zachyceny v publikované vědecké literatuře, avšak určité vztahy, obzvláště pak vliv vstupních argumentů, nebyl nikde příliš uspokojivě popsán. Právě na tyto problémy se tedy snaží můj program reagovat a věřím, že jeho výsledky budou pro praxi dále využitelné.

V této kapitole bude podrobně rozebrán algoritmus realizující segmentační metodu spektrálního shlukování. Bude ukázáno, jak jsem jej pro svou potřebu implementoval společně s vysvětlením použitých metod dodatečných knihoven, které byly popsány v předchozích kapitolách. Zaměřím se také na to, v čem má mnou navržená implementace rezervy a bude nastíněno, ve kterých oblastech se nachází prostor pro zlepšení a další vývoj pro dosažení lepších výsledků.

5.1 Hlavní myšlenka

Na začátku mého algoritmu je nutno vybrat vstupní obrázek, se kterým se bude v programu dále pracovat. Tento výběr se realizuje skrze vstup z klávesnice výběrem odpovídající položky menu, které se zobrazí při spuštění programu. Program jsem takto pojal z toho důvodu, aby nebylo nutno jej spouštět znovu od začátku pro každý výpočet nad daným obrázkem. Tímto způsobem se dají snáze provádět experimenty nad vypočtenými hodnotami daného vstupního obrázku bez nutnosti přerušení a opětovného zadávání všech vstupních hodnot včetně jména souboru obrázku.



7 – Ukázka hlavního menu mého programu

Po vybrání obrázku máme v programu několik možností. Nemáme-li zatím vypočteny pro námi nahraný obrázek žádný set vlastních čísel s jejich vlastními vektory, zvolíme z hlavního menu položku pro jejich výpočet. Program si poté vyžádá ruční zadání vstupních parametrů pro metodu spektrální segmentace, kterými jsou:

1. Počet požadovaných vlastních čísel s jejich vlastními vektory
2. Maximální počet výpočetních iterací pro knihovny ARPACKu
3. Hodnotu σ pro výpočet Gaussovy funkce při porovnávání pixelů
4. Požadované jméno výstupního souboru

První argument, kdy zadáváme požadovaný počet vlastních čísel s jejich vektory, je víceméně jasný. Hodnota, kterou zde zadáme, však znatelně ovlivňuje dobu běhu celého výpočtu. Konkrétnější informace o tom, jaký vliv má volba počtu požadovaných vlastních čísel s jejich vektory, budou podrobněji popsány v samostatné podkapitole.

Maximální počet výpočetních iterací také ovlivňuje dobu běhu výpočtu. ARPACK se totiž k výsledku rozkladu vstupní matice na vlastní čísla s jejich vektory přibližuje postupně v jednotlivých iteracích, což je základ jeho metody. Zadáním maximálního počtu těchto iterací tedy nastavujeme, do jaké hloubky se má požadovaný výsledek počítat, jedná-li se o složitější vstupní problém. Jak bude vysvětleno v dalších kapitolách, některé problémy při zadání určitých vstupních parametrů potřebují pouze pár iterací, zatímco jiné podstatně více. Právě odhalení těchto netriviálních závislostí bylo cílem mé práce.

Hodnota σ pro výpočet Gaussovy funkce prakticky určuje váhu, kterou budou mít jednotlivé diference jasů či barev pixelů na výsledek. Jedná se také o jeden z nejdůležitějších vstupních parametrů, který značně ovlivňuje jak dobu výpočtu, tak kvalitu výsledné segmentace. Volba této hodnoty není snadná a intuitivní a proto jí rovněž bude věnován samostatný úsek mé práce, kde bude popsána na jednotlivých experimentech.

Požadované jméno výstupního souboru se do programu zadává pro účely uložení vypočtených vlastních čísel s jejich vlastními vektory na pevný disk. Jelikož je výpočet větších vstupních obrázků mnohdy velmi zdlouhavý, zvláště pak s určitým nastavením vstupních argumentů, je uložení výsledných hodnot pro pozdější zpracování velmi výhodné. V mém programu je pak takto uložený soubor možno načíst a přiřadit vstupnímu obrázku bez potřeby hodnoty znovu počítat. Tímto způsobem lze efektivněji provádět následné experimentování s finálním shlukováním a sledovat tak vliv volby počtu hledaných segmentů na výsledek. Uložený soubor obsahuje informace o počtu a velikosti vypočtených vlastních čísel a jejich vektorů, takže při jeho pozdějším načtení není nutno znovu tyto údaje zadávat.

Pokud výpočet vlastních čísel s jejich vektory proběhne v pořádku, následuje metoda, ve které dojde k jejich přepočtu. Vektory budou normalizovány na jednotkovou délku a výsledky se uloží do souboru na disk. Je také možno pro potřeby vlastní kontroly zvolit i dodatečné uložení výsledků do textového souboru, který je formátován tak, aby byl snadno čitelný pro člověka.

5.2 Kontrola výpočtu

Pro kontrolu správnosti výpočtu jsem do programu implementoval ověření vztahu, dle kterého se vlastní čísla a jejich vlastní vektory knihovnou ARPACK počítají. Metoda EquationCheck tak pro všechna vypočtená vlastní čísla a jejich vektory ověří, jakou přesnost jednotlivé výsledky dosáhly. Kontrola se provádí násobením a sečtením matic dle následujícího vztahu 8.

$$Lx - lx = 0$$

8 – Rovnice pro výpočet vlastních čísel s vlastními vektory

Výsledky pro jednotlivá vlastní čísla a jejich vektory by ideálně měly být nulové. Výsledné hodnoty Sum pak reprezentují celkovou odchylku, která se v průběhu kontroly jednotlivých řádků sečte z rozdílů pravé a levé strany rovnice. Ukázku této kontroly demonstruje následující obrázek 9 pořízený z běhu programu. První dvě vypočtené hodnoty jsou irelevantní, protože odpovídají vlastnímu číslu hodnoty 0, které ze vztahu vyřadí druhý člen pravé strany, a tudíž dojde k nasčítání přílišné odchylky. Jak je vidět z obrázku, odchylky takového řádu se opravdu vyskytují pouze u vlastních čísel s hodnotou 0. Pro ostatní vypočtená vlastní čísla a jejich vektory jsou nasčítané odchylky velmi malé, což znamená, že byly vypočteny co možná nejlépe.

```
time =    0,420 s
eigenvalue  1: -0,000000000000000
eigenvalue  2: 0,000000000000000
eigenvalue  3: 0,000000000000007
eigenvalue  4: 0,001387003477261
eigenvalue  5: 0,001980195446388
eigenvalue  6: 0,002495364171202
eigenvalue  7: 0,004594289972181
eigenvalue  8: 0,007716867748271
eigenvalue  9: 0,010645837434811
eigenvalue 10: 0,011980132492468

Eigenvalues and eigenvectors written to a HR file

Beginning of equation check
  -1. equation checked. Sum: 129,54365205
  -2. equation checked. Sum: 22,58306070
  -3. equation checked. Sum: -0,14992193
  -4. equation checked. Sum: -0,79557499
  -5. equation checked. Sum: 2,06742670
  -6. equation checked. Sum: 1,34292540
  -7. equation checked. Sum: 1,03526478
  -8. equation checked. Sum: 0,46227094
  -9. equation checked. Sum: 0,55932035
  -10. equation checked. Sum: -0,84207439
Eigenvectors normalized
Eigenvalues and eigenvectors successfully computed...
Saving -> Eigenvalues and eigenvectors saved to file 31.egf
```

9 – Ukázka zpětné kontroly výpočtu vlastních čísel a jejich vektorů

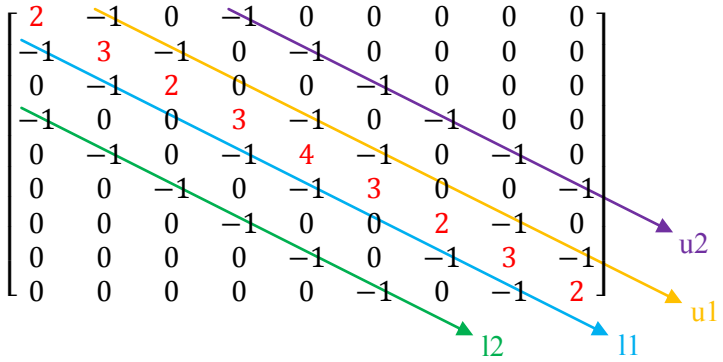
5.3 Efektivita programu

Vzhledem ke způsobu jakým je konstruována Laplaceova matice, by její uložení ve standardním formátu zabíralo příliš mnoho místa. V případě, že bychom měli vstupní obrázek o výšce i šířce 30 pixelů, by bylo nutné pro standardní uložení odpovídající Laplaceovy matice alokovat v paměti pole floatů o velikosti 810 000 prvků, což by v operační paměti zabralo 3,24 MB. Laplaceova matice obecně dovoluje vyjádřit vztah mezi všemi pixely obrazu mezi sebou, tudíž se její prostorová náročnost dá vyjádřit následujícím vztahem 10, kde a a b odpovídají velikostem stran vstupního obrázku.

$$\Theta_p = (a \cdot b)^2$$

10 – Paměťová náročnost pro standardní způsob dvojrozměrného uložení

Pro potřeby algoritmu spektrálního shlukování však není nutno uchovávat všechny hodnoty Laplaceovy matice. Ze vstupního obrázku do ní uvádíme pouze 4 hodnoty pro každý pixel, které odpovídají podobnosti s jeho čtyřmi sousedy. Dále pak na hlavní diagonálu této matice ukládáme součet daného řádku s opačným znaménkem. Na následujícím obrázku 11 je znázorněno, jak může vypadat Laplaceova matice pro vstupní obrázek velikosti 3×3 pixely, kde všechny mají stejnou barvu.



11 – Ukázka Laplaceovy matice pro obrázek o velikosti 3×3 pixely

Jak je z uvedeného příkladu vidět, nenulové hodnoty se mohou vyskytovat pouze na hlavní diagonále, či na čtyřech sub-diagonálách u1, u2, l1, nebo l2. Jednotlivé sub-diagonály odpovídají čtyřem sousedním pixelům takto:

- l2 – spodní soused
- l1 – levý soused
- u1 – pravý soused
- u2 – horní soused

Zatímco sub-diagonály l1 a u1 budou vždy těsně přimknuty k hlavní diagonále, vzdálenost sub-diagonál l2 a u2 se bude měnit v závislosti na šířce vstupního obrázku. Od prvního prvku budou vždy vzdáleny přesně o tuto šířku, čímž lze snadno vyjádřit jejich velikost v rámci celé matice. Jak je možno na ukázce vidět, sub-diagonály l1 a u1 mají na každé pozici odpovídající šířce vstupního

obrázku nuly. Tyto nulové hodnoty se v nich vyskytují z toho důvodu, že každý poslední pixel řádku vstupního obrazu nemůže mít pravého souseda a každý první pixel řádku souseda levého. Můj program těchto vědomostí využívá a pro ukládání hodnot Laplaceovy matice si vytvoří pět pomocných polí, které odpovídají hlavní diagonále a zmíněným čtyřem sub-diagonálám. Do nich ukládá hodnoty, které by jinak umisťoval do standardního dvojrozměrného provedení v paměti. Tímto přístupem jsem byl schopen maximálně zmenšit paměťovou náročnost, kterou toto zadávání potřebuje pro svůj běh. Například pro vstupní obrázek velikosti 100×100 pixelů zabere tento způsob uložení pouze 12% místa, které by bylo nutno vyhradit pro standardní dvojrozměrné uložení. Tuto optimalizaci bylo nutné udělat i z toho důvodu, že paměťové nároky standardního dvojrozměrného uložení by ani neumožňovaly pracovat s reálnými vstupními obrázky velikosti například 800×600 pixelů.

5.4 Práce s knihovnami

Jak již bylo zmíněno v předchozích kapitolách, pro zjednodušení práce a získání lepších výsledků jsem ve svém programu využil funkcí dvou veřejně dostupných knihoven. V následujících podkapitolách tedy budou k vidění ukázky kódu a vysvětlení prvků, které jsem z pestré nabídky těchto knihoven využil. Než tedy přejdu k popisu a vysvětlení jednotlivých použitých funkcí, rád bych zmínil jeden problém, na který jsem narazil na samém začátku mé práce. Když jsem do projektu vkládal příkazy pro zahrnutí (include) knihoven do svého kódu, překladač vypsal mnoho chyb, kterým jsem zcela nerozuměl ani přes vyhledání jejich významu na internetu. Po vysilujícím experimentování s nastavením projektu a překladačů jsem nakonec jednoduše změnil pořadí, ve kterém jsem tyto knihovny do projektu zahrnoval, což tento problém vyřešilo. Toto překvapivé řešení, se kterým jsem se doposud nesetkal, jsem tudíž pro případné další zájemce o práci s těmito knihovnami nechtěl opomenout. Knihovny ARPACKu, které jsem ve svém programu využil, by měly zřejmě být jiné, avšak hlouběji o tomto problému bude pojednáno v podkapitole o problémech. Funkční posloupnost zahrnutí knihoven do programu nakonec vypadala takto:

```
#include <cstdlib>

// OpenCV knihovny
#include <highgui.h>
#include <cxcore.h>
#include <cv.h>

// Arpack++ knihovny
#include "areig.h"
```

12 – Ukázka funkční posloupnosti zahrnutí knihoven

5.4.1 Práce s OpenCV

S knihovnami OpenCV jsem pracoval převážně z důvodu snadné manipulace s daty vstupních a výstupních obrázků. Za dobu svého studia jsem se naučil využívat makra, které tyto knihovny zpřístupňují, a která mají za následek velmi efektivní a hlavně rychlý přístup k datům. Díky tomu jsem nemusel implementovat žádné své nové funkce, které by toto umožňovaly, a mohl tak svou práci urychlit a zkvalitnit. Následující dvě ukázky kódu z mého programu předvádí, jakým způsobem jsem ve své práci přistupoval k datům obrázku. Tímto způsobem se můžeme dostat k jednotlivým složkám obrazu a to až na úroveň jednotlivých sub-pixelů, tedy nejčastěji barevným složkám RGB. Následující

ukázka 13 znázorňuje, jak jsem konkrétně přistupoval k pixelům obrázku, který jsem předtím převedl na jednokanálový, to jest obsahující pouze jasovou informaci. Indexové proměnné zde tedy určují pouze polohu nabíraného, či zapisovaného pixelu.

```
// Nabrani hodnot 4-sousednosti z obrazku do promennych
if (y-1>=0) a = CV_IMAGE_ELEM(vstup3,double,y-1,x); // Nahoru
if (x!=0)   b = CV_IMAGE_ELEM(vstup3,double,y,x-1); // Doleva
if (y+1<Y)  c = CV_IMAGE_ELEM(vstup3,double,y+1,x); // Dolu
if (x!=X-1) d = CV_IMAGE_ELEM(vstup3,double,y,x+1); // Doprava
e = CV_IMAGE_ELEM(vstup3,double,y,x); // Stred
```

13 – Ukázka práce s knihovnami OpenCV

Jak bylo řečeno, stejným makrem lze také přistupovat ke složkám barevného obrazu. V tomto případě si však musíme dávat na indexování větší pozor, abychom nezapisovali na špatnou část paměti, která už danému pixelu, či dokonce obrázku, nepatří. Následující ukázka 14 tedy ukazuje, jak přistupovat k jednotlivým sub-pixelům po jednotlivých složkách.

```
double eb = CV_IMAGE_ELEM(vstup,double,y,x);
double eg = CV_IMAGE_ELEM(vstup,double,y,x+1);
double er = CV_IMAGE_ELEM(vstup,double,y,x+2);
```

14 – Ukázka nabírání jednotlivých sub-pixelů pomocí maker OpenCV

Zde hodnoty indexu osy x znamenají přístup k jednotlivým složkám, které jsou uspořádány pozpátku v posloupnosti BGR. Přičtením jedničky či dvojky se tedy stále pohybujeme v rámci jednoho pixelu a nastavujeme zvlášť jednotlivé hodnoty sub-pixelů. Ve složitějších vzorcích je tedy potřeba na tento fakt dát pozor, abychom se vyvarovali zápisovým problémům mimo aktuální buňku či paměť, které překladač nenahlásí jako chybu.

Dále se mi v práci velmi hodily funkce pro práci s grafickými okny, ve kterých jsem schopen zobrazovat jakékoli obrazové informace. Díky intuitivním příkazům jsem snadno vytvořil jednotlivá okna, se kterými jsem dále pracoval. Pro lepší zobrazení je možno jednoduše zvětšit malé obrázky z důvodu snazší kontroly uživatelem. Následující ukázka kódu 15 předvádí, jakým způsobem jsem nahrál obrázek z pevného disku, vytvořil okno pro zobrazení, nastavil vlastní pevnou velikost, přiřadil oknu daný obrázek a po stisku klávesy uživatelem okno uzavřel.

```
IplImage *img = cvLoadImage("Obrazek.jpg",1);
cvNamedWindow("Vstup",0);
cvResizeWindow("Vstup",200,200);
cvShowImage("Vstup",img);
cvWaitKey(0);
cvDestroyWindow("Vstup");
```

15 – Ukázka práce s okny pomocí knihoven OpenCV

5.4.2 Práce s ARPACK++

Knihovny ARPACKu poskytovaly mému programu hlavní oporu. Jelikož je algoritmus spektrálního shlukování přímo založen na nalezení správných vlastních čísel a jejich vektorů pro vstupní Laplaceovu matici, bylo využití těchto knihoven ústředním motivem mé práce. Prvním a zároveň největším úskalím pro snadnou práci s funkcemi těchto knihoven pro mě bylo pochopit a navrhnout implementaci pro algoritmus, který se stará o zformátování vstupních dat. Knihovny ARPACKu totiž definují konkrétní struktury, které je nutno využít při zadávání vstupních matic. Jedná se o formáty:

- Formát hustě obsazeného vektoru (dense vector)
- Formát komprimovaných řídkých sloupců (CSC – Compressed Sparse Column)
- Pásový formát (band format)

Každý z nich má jasně definovaná pravidla pro svou konstrukci. Pro účely mého programu jsem využil druhou možnost z výše uvedených, tedy formát komprimovaných řídkých sloupců (dále jako CSC formát). Tento formát zabírá v paměti nejméně místa, a proto dále rozšiřuje efektivitu mého předchozího systému úspory místa v paměti. Do CSC formátu se ukládají data pomocí tří hlavních polí.

1. **Values** – V tomto poli jsou za sebou naskládány všechny nenulové hodnoty vstupní Laplaceovy matice. Výsledkem je tedy značně dlouhé pole floatů.
2. **Irow** – Toto pole celých čísel (integer) představuje indexy sloupců, ve kterých se uložené hodnoty z pole values vyskytují. Díky tomuto poli jsou metody schopny jednoduše zjistit pozici prvku v matici i bez její standardní dvojrozměrné struktury.
3. **Pcol** – Toto pole celých čísel (integer) ukládá sečtený počet prvků na jednotlivých řádcích začínající od nuly. Rozdílem dvou sousedních hodnot z tohoto pole tedy metoda zjistí, kolik nenulových hodnot se na daném řádku vstupní matice nachází, aby byla schopna správně interpretovat indexy z pole Irow.

Matice, se kterou ve svém programu pracuji, je symetrická. V takovém případě funkcím ARPACKu stačí do předem definovaného formátu symetrické matice uložit pouze její horní, či spodní trojúhelníkovou oblast pro další snížení paměťových nároků. Formátovací algoritmus, který matici transformuje do této nové struktury polí, začíná od levého horního rohu matice a postupuje směrem doprava po řádkových pozicích vybrané trojúhelníkové oblasti. V mém případě ukládám pro další zpracování pouze horní trojúhelníkovou oblast včetně hlavní diagonály. Jelikož se nejedná o zcela intuitivní systém zadávání vstupu, následující ukázka 16 bude sloužit pro lepší demonstraci a případné pochopení takového vstupního formátování. Následující ukázková matice ve standardním formátu bude tedy odpovídat třem polím pod ní.

$$\begin{bmatrix} 2 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 3 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 3 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{bmatrix}$$

16 – Příklad změny reprezentace Laplaceovy matice do CSC formátu

- Values: 2 -1 -1 3 -1 -1 2 -1 3 -1 -1 4 -1 -1 3 -1 2 -1 3 -1 2
- Irow: 0 1 3 1 2 4 2 5 3 4 6 4 5 7 5 8 6 7 7 8 8
- Pcol: 0 3 6 8 11 14 16 18 20 21

Poté, co získáme vstup například v tomto formátu, je potřeba jej předat náležitému objektu, který bude pro výpočetní metody ARPACKu určovat jeho strukturu. V mém případě se tedy jednalo o symetrickou matici. Pro výpočet vlastních čísel jsem se rozhodl použít jednu z nejjednodušších knihoven balíku ARPACK, knihovnu SuperLU, která pro mé účely bohatě stačila. V následující ukázce z mého programového kódu tedy na prvním řádku definuji takovou matici, která ve svém názvu obsahuje lu, což odkazuje právě na použití knihoven SuperLU. Použití funkcí knihoven SuperLU je zahrnuto v hlavičkovém souboru Areig.h, jenž byl zmíněn v jedné z předchozích kapitol. Jako parametry matice tedy předáme její dimenzi, počet nenulových hodnot a tři pole získané z minulé ukázky. Dále je pak nutno definovat typ problému, který určí strategii výpočtu. V mém konkrétním případě jsem vždy počítal symetrický standardní problém vlastních čísel a jejich vektorů, což reflektuje i název třídy na druhém řádku. Prvním vstupním parametrem této třídy je požadovaný počet vlastních čísel a jejich vektorů, které chceme po ARPACKu vypočítat. Dalšími jsou pak samotná matice definována o řádek výše a řetězec, který určuje, zda chceme obdržet hodnoty nejvyšší, či nejmenší. Pro potřeby následujícího shlukování jsem ve svém programu využíval vždy vlastní čísla a jejich vektory začínaje od nejmenších, tudíž argument „SM“ jako smallest.

```
ARluSymMatrix<double> matrix( dimension, nnz, values, irow, pcol);
ARluSymStdEig<double> problem( numEigenVals, matrix, "SM" );
```

17 – Ukázka zadávání vstupu pro zpracování knihovnami ARPACK

Třídě problem se poté zavolá metoda FindEigenvectors(), která spustí výpočet. Po skončení výpočtu a korektním nalezení požadovaného řešení se výsledky musí extrahovat opět z této třídy. Pomocí cyklu již poté není těžké využít metody této třídy pro navrácení všech vypočtených vlastních čísel a jejich vektorů. Pro další zpracování je ve svém programu ukládám do jednorozměrných polí. Tyto pole pak společně s dalšími informacemi o vstupním obrázku ukládám na disk pro možnost pozdějšího zpracování.

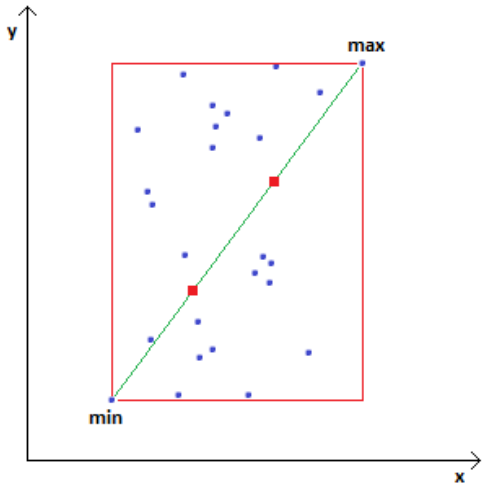
5.5 Shlukování K-means

Posledním krokem po získání výsledných vlastních čísel a jejich vektorů je shlukování, které určí jednotlivé segmenty. Nejprve se však výsledky z předchozího kroku přepočtou na nové souřadnice, jak bylo zmíněno v jedné z předchozích kapitol. Tyto nové souřadnice k původním bodům vstupního obrazu přidávají další vlastnosti, které umožňují jejich snazší segmentaci. Je však potřeba zvolit nějaký shlukovací algoritmus, který dovede provést segmentaci těchto nových souřadnic. Jak bylo zmíněno v textu dříve, dimenze prostoru, ve kterém budeme toto finální shlukování provádět, je dána počtem vypočtených vlastních čísel a vektorů. Vzhledem k tomu, že v literatuře, ze které jsem informace o této metodě čerpal, zmiňovali autoři algoritmus k-means jako referenční pro začátek, zvolil jsem právě jej.

Jako vstup pro tento algoritmus dávám nové souřadnice vstupních bodů ve formě přepočtených vlastních vektorů, jejich velikost a počet, pole pro uložení informací o náležitosti jednotlivých bodů k nově nalezeným segmentům, počet požadovaných segmentů k nalezení a maximální počet iterací. Algoritmu je totiž dopředu nutno zadat, na kolik oblastí má vstupní body rozdělit, jelikož toto neumí zjistit sám. Zadáním počtu požadovaných segmentů tedy zásadně ovlivňujeme výsledek, který nám algoritmus k-means vrátí. Vliv toho počtu na výsledek bude podrobněji k vidění v dalších kapitolách věnovaným experimentům. Maximální počet iterací pak nastavuje dovolenou hranici, kdy má algoritmus své hledání oblastí ukončit. Shlukování pomocí k-means se totiž ze začátečního stavu snaží k výsledku dojít po jednotlivých krocích.

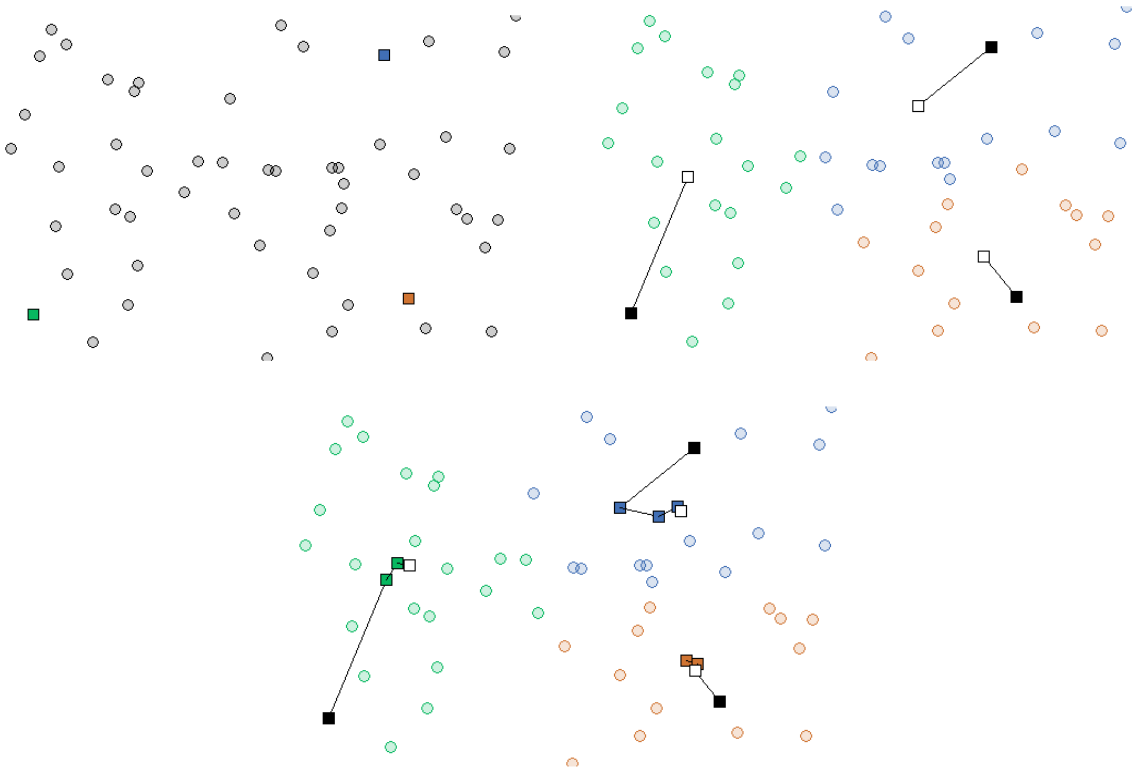
Počáteční stav algoritmu k-means může také mít nezanedbatelný vliv na výsledek. Rozložení počátečních bodů lze realizovat několik způsoby. Prvním zmíněným způsobem je náhodná inicializace počátečních pozic. Tento způsob je v literatuře, ze které jsem čerpal, zmíněn jako první. Jelikož je však náhodnost dle mého málo exaktní věc, na které by se daly stavět relevantní a porovnatelné výsledky, rozhodl jsem se pro druhý způsob, kde jsou počáteční souřadnice určeny pevně dle určitého systému. Do svého programu jsem v rámci metody k-means však pro možnost volby zakomponoval obě dvě tyto metody. Z počátku jsem nejvíce využíval metodu s pevným stanovením počátečních bodů. Ta najde ze vstupních souřadnic bodů jejich maximum a minimum a rozmístí počáteční body rovnoměrně mezi ně.

Na následujícím obrázku 18 je uveden dvojrozměrný příklad mé metody pro určení dvou počátečních bodů (červeně), která body rovnoměrně rozprostře do oblasti se vstupními body (modře). Druhá metoda, která body určuje náhodně, také nabere ze vstupu krajní hodnoty, ale počáteční body v rámci nich určí náhodně. Tuto metodu s náhodným určením počátků jsem používal především v pozdějších fázích a experimentech, kterým je věnována celá kapitola.



18 – Pevné určení počátečních bodů pro finální shlukovací metodu k-means

Po určení počátečních bodů, ze kterých bude výpočet dále vycházet, algoritmus určí, které vstupní body jsou daným počátečním polohám nejblíže. Po přiřazení každého vstupního bodu k jednomu z uzlů se vypočte těžiště aktuálního shluku. Do této nově vypočtené pozice se uzel přesune a algoritmus se opakuje znovu od měření vzdáleností k přiřazování. Shlukování pak můžeme ukončit za dvou podmínek. První z nich je, že se uzly přestanou hýbat, nebo je jejich posun menší než předem stanovená hodnota. Druhou možností pak je, že shlukování dosáhne maximálního počtu povolených iterací, které jsme mu nastavili na začátku jako vstupní parametr. Následující ukázka 19 demonstruje pohyb uzlů do nově vypočtených těžišť a rozdělování vstupních bodů do jednotlivých oblastí, které jsou značeny barevně. Úsečkami jsou v ukázce vyznačeny cesty, kterými se jednotlivé uzly pohybovaly do svých nových pozic.



19 – Ukázka postupu segmentace algoritmu k-means

5.6 Problémy

V průběhu mé práce se v rámci programu objevily určité problémy, o kterých bude pojednávat tato kapitola. Zpočátku se jednalo hlavně o problémy se zprovozněním knihoven ARPACKu a OpenCV tak, aby fungovaly v rámci jednoho projektu. Nenašel jsem žádné dostupné příklady, které by se tomuto problému konkrétně věnovaly. Především díky komplexnosti knihoven ARPACKu a jejich provázanosti jsem nebyl schopen korektně projekt nastavit tak, aby využíval nainstalované knihovny z Unixové složky k tomu určené. V začátcích jsem si vystačil s využitím knihoven, které byly přiloženy v rámci jednotlivých ukázek funkcionality balíku ARPACK++. Postupným odebíráním zahrnutých knihoven a hlavně dalšími úpravami programového kódu jsem počet knihoven zredukoval na pouhou jednu základní, která mi poskytuje potřebné metody. Konkrétně se jedná o knihovnu Areig.h, kterou popisuji výše v jedné z předchozích kapitol. Myslím tedy, že tato problematika by zasloužila případným dalším řešitelem mého problému větší pozornost.

Zajímavým problémem, se kterým se občas stále setkávám, je reakce oken knihoven OpenCV na vykreslení zadaného vstupu. Občas se při zobrazení větších obrázků stává, že při finálním shlukování a hledání konečné segmentace okno jednoduše přestane reagovat a výstup se v něm nepřekreslí na aktuální. Stačí však programu nechat nalézt dalších několik výsledků, což okno aktualizuje, a za chvíli okno zase reagovat začne. Nepodařilo se mi přijít na to, čím je tento problém způsoben.

V programu se mi občas stávalo také to, že program spadl na výjimce hlásící neoprávněný přístup do paměti při dealokaci dynamicky vytvářených polí příkazem delete. Po určitých změnách v programu,

které se týkaly hlavně manipulace s okny knihoven OpenCV, tato dealokace přestala fungovat, ačkoli pole i ukazatele na něj zůstaly zcela stejné a nezměněné, o čemž jsem se přesvědčil pomocí debuggeru. Jedná se nedůležitý problém, se kterým si poradí sám operační systém, a který negativně neovlivňuje chod mého programu. Přesto jsem jej však chtěl zmínit pro případné pokračovatele v mé práci.

5.7 Prostor pro zlepšení a vývoj

V této kapitole se zaměřím na popis oblastí mého programu, které by se daly v budoucnu vylepšit, aby program dosahoval lepších výsledků. Jedná se především o oblasti, na jejichž vylepšení mi nezbyl čas v rámci vývoje, a které se netýkají opravy problémů zmíněných v předchozí kapitole.

Jako první věc, která mě napadá po delší době práce s mým programem, bych rád uvedl časovou náročnost výpočtu pro větší reálné vstupní obrázky. Jelikož segmentační algoritmy obecně chceme aplikovat spíše na reálné snímky získané v praxi, je jasné, že uměle vytvořené obrázky s jasnými segmenty často zpracovávat nebudeme. Pro využití tohoto algoritmu v nějaké aplikaci by tak jistě bylo potřeba, aby byl schopen výsledek nalézt dříve než za několik minut, jak je to mu pro větší vstupy nyní. Zlepšení stávajícího stavu by tedy mohlo být provedeno například předěláním výpočtu vlastních čísel a jejich vektorů pomocí paralelně pracující verze knihoven s názvem PARPACK, které jsem okrajově zmiňoval v jedné z předchozích kapitol. Věřím, že kdyby byl schopen ARPACK ke své práci využít všechna dostupná jádra procesoru, byl by výpočet znatelně rychlejší. Vzhledem k počtu fyzicky samostatných jader u dnešních procesorů dostupných i pro běžný trh jsem si jist, že by tato možnost byla velmi zajímavá na vyzkoušení.

Další vylepšení, které by dle mého názoru znatelně zlepšilo především kvalitu výsledku, by se mohlo týkat implementace komplexnějšího algoritmu pro finální shlukování nových souřadnic, které dostaneme po přepočtu výsledných vlastních vektorů. Mnou implementovaný algoritmus k-means segmentaci zvládá a jeho výsledky nejsou špatné, avšak jeho rezervy jsou v experimentech znatelné. Z množství vstupních obrázků, které jsem programem zpracoval, jsem nabyl dojmu, že informace pro správné nalezení kvalitnějších segmentů se v nově vypočtených souřadnicích bodů nacházejí, ale můj algoritmus je prostě neumí správně vyhodnotit. Mnohdy se mi totiž podařilo nalézt znatelně kvalitnější části obrázku, když jsem používal náhodné rozmístění počátečních uzlů ve svém algoritmu k-means. Tato náhodná metoda však nikdy nebyla schopna správně vyhodnotit segmenty celého obrázku, nýbrž pouze nalézt pěknější části daného celku. Z prostudování příkladů, které se týkají například shlukování pomocí mixu gaussiánů, jsem pochopil, že by stálo za to podobnou metodu aplikovat i v mém programu. Díky jeho komplexnosti jsem však již v závěru své práce, kdy jsem se k těmto informacím dostal, neměl dostatek času pro jeho implementaci a otestování v rámci svého programu. Věřím však, že by podobné vylepšení posunulo mou práci zase o krok dále.

Jisté vylepšení by si zasloužila i struktura mého programu. Vzhledem k časové náročnosti výpočtu vlastních čísel a jejich vektorů by se v programu hodilo mít možnost například dávkového zpracování vstupních obrázků pro pozdější analýzu. Tím by odpadla nutnost pro každé hledání nových vlastních čísel a jejich vektorů znovu programu zadávat hodnoty vstupních argumentů. Tyto vstupy by mohly být obsaženy například v předem připraveném souboru, který by zpracování řídil a který by programu

vždy předložil odpovídající vstupní obrázek společně s nastavením parametrů pro jeho zpracování. Program by tak mohl běžet nepřetržitě po delší dobu tehdy, když bychom počítač zrovna nevyužívali, či například na stroji určeném primárně k tomuto využití. Výsledky by byly uloženy na pevném disku a tím by je později bylo možné analyzovat hromadně, což by pro získávání nových poznatků o tomto algoritmu z jeho experimentů bylo jistě přínosné.

6 Dosažené výsledky

Tato hlavní kapitola bude popisovat výsledky, ke kterým jsem se dostal s použitím svého algoritmu. Jelikož pro nastavení metody spektrálního shlukování je nutno alespoň určitým způsobem porozumět zákonitostem kolem volby hodnot vstupních parametrů, bude se tato kapitola věnovat hlavně popisu a rozboru výsledků vzhledem k tomuto nastavení. Jako první budou předvedeny výsledky segmentace pro různé typy vstupních obrázků, aby bylo vidět, jak dobře si tento algoritmus dokáže poradit se segmentací obrazu. K těmto obrázkům bude k dispozici komentář, ve kterém nastíním, jakou obtížnost mělo nalezení uvedených nejlepších výsledků. V další podkapitole se zaměřím na rozbor vlivu jednotlivých vstupních argumentů na dobu běhu hlavního výpočtu programu, tedy výpočtu vlastních čísel a jejich vektorů. Závislost, kterou se doba běhu řídí, totiž není triviální a mohu říci, že jí stále plně nerozumím. Na jednotlivých experimentech jsem však získal alespoň jakousi povrchní intuici, kterou jsem se poté v projektu dále řídil. Poslední podkapitola pak bude pojednávat o vlivu vstupních parametrů na kvalitu výsledné segmentace. Ta je totiž zásadně ovlivněna tím, jak správně zvolíme tyto vstupní parametry, pro jejichž volbu neexistuje žádný exaktní vzorec.

6.1 Ukázky segmentace

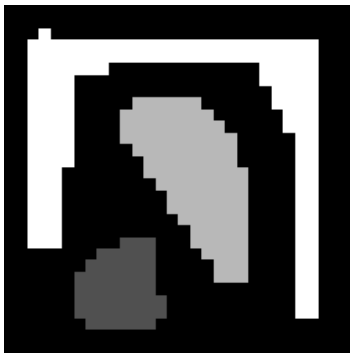
V této kapitole budou ukázány výsledky mého algoritmu pro jednotlivé typy vstupních obrázků. Ukázky, které zde budou k vidění, představují obvykle ten nejlepší segmentační výsledek, ke kterému jsem se byl schopen v rámci svého programu dopracovat. Jako první kategorii jsem zvolil umělé obrázky. Na těchto jednoduchých vstupech jsem od začátku vývoje testoval funkčnost svého programu a ověřoval si na nich jeho zlepšující se schopnost vracet správné výsledky. Postupně jsem došel až ke složitějším umělým obrázkům, které byly téměř na hranici možností mého algoritmu. Druhou kategorii jsem věnoval obrázkům reálným, se kterými se můžeme setkat v praktických aplikacích. Vstupní obrázky jsou proto nejčastěji výřezem nějaké reálné fotografie. Jako poslední kategorii jsem zvolil těžce segmentovatelné obrázky, se kterými je v praxi obvykle největší problém. Jedná se především o obrázky, ve kterých je přítomno velké množství šumu. Dále pak takové obrázky, ve kterých se vyskytují složité objekty či rušivé pozadí. V následujících odstavcích budu často zmiňovat referenční hodnoty, jejichž nastavení a popis však bude součástí podkapitoly o vlivu vstupních parametrů na výsledek. Hodnota σ, která je v následujícím textu mnohokrát zmíněna, patří vzorci pro stanovení podobnosti dle již zmíněného předpisu, který pro přehlednost znovu přikládám.

$$P_{i,j} = e^{-\frac{(v_i - v_j)^2}{2\sigma^2}}$$

20 – Vzorec pro podobnost pixelů, pro nějž nastavujeme parametr σ

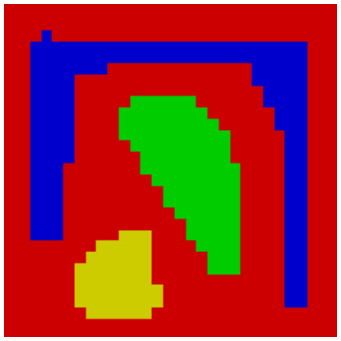
6.1.1 Umělé obrázky

Jako první obrázek, který jsem pro potřeby svého programu vytvořil, byl vstup se třemi segmenty na černém pozadí. Pro zkoumání vlivu vstupních parametrů jsem tyto segmenty odlišil různou hloubkou šedého odstínu. Jejich tvar a umístění jsem také volil záměrně tak, aby nebyly triviální. Proto je prostřední segment záměrně obklopen výraznější světlou oblastí seshora, zatímco zespod by mohl běžné jednoduché algoritmy mást segment tmavý. Vstupní obrázek je vidět na následující ukázce 21.



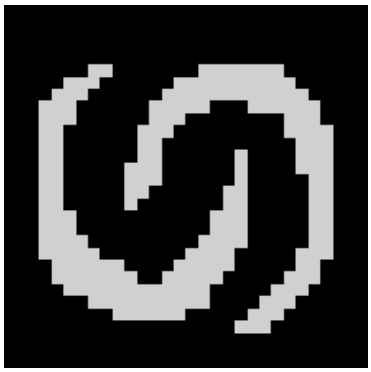
21 – První umělý vstupní obrázek

Tento obrázek má velikost 30×30 pixelů, což jsem ze začátku zvolil pro co nejrychlejší zpracování mým algoritmem, abych se byl schopen s programem efektivně naučit pracovat bez dlouhých prodlev způsobených výpočtem větších vstupů. Poté, co byl hlavní program hotov, jsem byl schopen správně určit jednotlivé oblasti tohoto obrázku velice brzy. Pro zadané referenční hodnoty, o kterých jsem se dozvěděl primárně z literatury o tomto tématu a vedoucího mé diplomové práce, se téměř podařilo správně určit všechny tři hlavní oblasti. Pro původní nastavení hodnot však algoritmus měl problémy s nalezením nejtmavějšího segmentu, který se nalézá ve spodní části obrázku. Program vždy určil prostřední dva oblé segmenty jako jeden celek stejnou barvou, pokud tedy do finálního shlukování spodní tmavý segment vůbec zahrnul. Po malém snížení hodnoty σ z původních 0,1 na 0,08 byl program hladce schopen rozpoznat všechny oblasti obrázku včetně správného jednolitého určení pozadí. Tento výsledek je znázorněn na dalším obrázku 22.



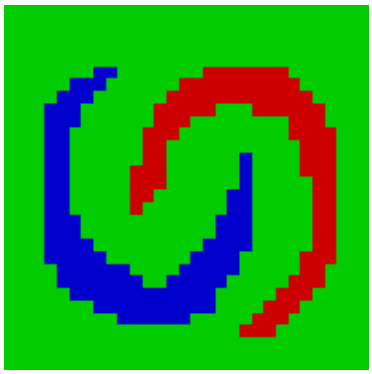
22 – Výsledek korektní segmentace prvního umělého vstupního obrázku

Jako další umělý obrázek, na kterém jsem si vyzkoušel funkci programu, jsem vybral dvě oblasti, které se vzájemně obklopují. Tento obrázek by byl jednoduchý pro určení triviálními metodami jako například prahováním, avšak metoda spektrálního shlukování funguje podstatně jinak, a proto jsem chtěl její schopnosti ověřit co možná nejlépe. Vstup v odstínech šedi je k vidění na následujícím obrázku 23.



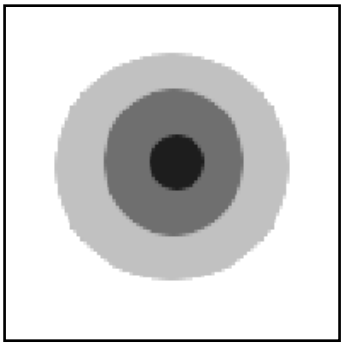
23 – Druhý umělý vstupní obrázek

Situace byla velmi podobná jako u předchozího příkladu. Program se pro referenční hodnotu σ vždy snažil určit šedé obloukové oblasti jako jeden celek. Po malém snížení hodnoty σ a výpočtu více vlastních čísel a jejich vektorů již však vždy určil oblasti správně. U tohoto obrázku bylo nutné zvolit správný počet vlastních čísel a jejich vektorů pro výpočet finálního shlukování tak, aby program vyhodnotil pozadí jako jeden celek. Pro více než 4 vlastní vektory totiž program většinou pozadí rozdělil na více oblastí, které se rozdělovaly uvnitř obloukových oblastí. Barevný výstup z programu s určenými a obarvenými oblastmi lze vidět na níže uvedeném obrázku 24.



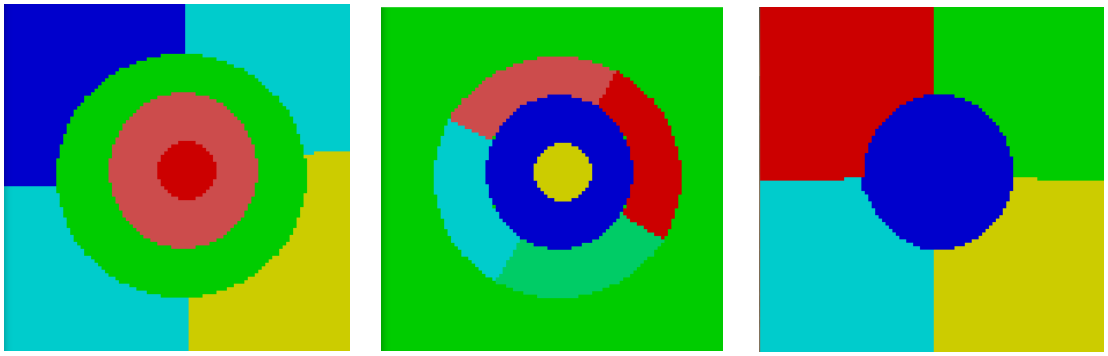
24 – Výsledek korektní segmentace druhého umělého vstupního obrázku

Dalším vstupem, se kterým již měl algoritmus znatelnější problémy, je obrázek se třemi kruhovými oblastmi, které se nacházejí uvnitř sebe s rozdílným jasem. Obrázek je oproti předchozím větší a jeho strana měří 100 pixelů. Tento umělý obrázek jsem vytvořil v posledních etapách testování svého programu, jelikož jsem z poznatků nabytých experimentováním věděl, že jeho správné určení na jednotlivé oblasti nebude zcela jednoduché. Vstupní obrázek v odstupňovaných odstínech šedi je znázorněn na další ukázce 25.



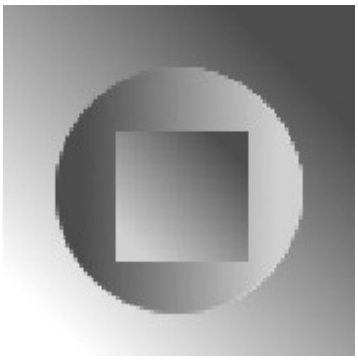
25 – Třetí umělý vstupní obrázek

Při zpracování mým programem za referenčních vstupních hodnot se mi nepodařilo správně určit dané oblasti obrázku. Níže uvedený obrázek napravo je nejlepším výsledkem programu, který pracoval blízko referenčních hodnot vstupních parametrů. Po finálním shlukování dovedl algoritmus správně určit jen prostřední šedou kruhovou oblast a i tato byla pro nalezení problematická. K výsledku napravo jsem se tak dostal mnohonásobným opakováním finálního shlukování s náhodně stanovenými počátečními body. Prostřední níže uvedený obrázek zachycuje výsledek, ke kterému jsem došel poté, co jsem snížil hodnotu parametru σ z původní 0,1 na 0,06. Tady se už programu lépe podařilo určit jednotlivé oblasti. Nebyl však nikdy schopen určit krajní kruhovou oblast jako jeden celek, což může také být zaviněno metodou finálního shlukování k-means. Pouze u tohoto výsledku však byl program schopen určit pozadí jako jeden celek, oproti krajním dvěma případům. Nejlepší výsledek se mi podařilo dosáhnout při razantním snížení hodnoty σ až na 0,04. Výpočet trval pětkrát déle, ale kvalita segmentace stoupla a bylo konečně možno určit kruhové oblasti zcela správně. Avšak ani po delším experimentování se mi nepodařilo volbou počtu vlastních čísel a jejich vektorů správně určit pozadí jako jeden celek. Obrázek nalevo je tak nejlepším výsledkem, který jsem pro tento vstup obdržel.



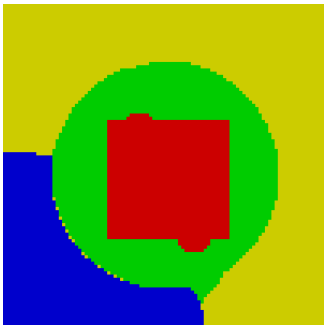
26 – Ukázky výsledků segmentace třetího umělého obrázku

Jako nejtěžší umělý vstupní obrázek pro segmentaci jsem vytvořil příklad, ve kterém se nachází postupný jasový přechod společně s kombinací oblastí různého tvaru. Vzhledem k tomu, že algoritmus měl vždy největší problém detekovat nejasné hrany, jsem věděl již dopředu, že tento problém bude pro můj program obtížné vyřešit. Hrany jednotlivých oblastí totiž díky tomu, že jsou z jasových přechodů, splývají. Samotné oblasti pro nalezení jsou pak také z těchto přechodů, jak je vidět na následujícím obrázku 27. Obrázek měl také velikost 100×100 pixelů, jako předchozí příklad.



27 – Čtvrtý umělý vstupní obrázek s jasovými přechody

Z počátku, kdy jsem zkoušel spouštět program s referenčními vstupními parametry, se mi nedařilo nalézt žádný vhodný výsledek, který by alespoň nastiňoval polohu a tvar jednotlivých oblastí. Jako u minulých příkladů bylo nutné snižovat hodnotu σ, což prodlužovalo čas potřebný pro výpočet. Kvalita výsledné segmentace za použití takto vypočtených vlastních vektorů se sice zvyšovala, ale pouze po určitou hranici. Minimální hodnotou σ, pro kterou byl výsledek použitelný a zároveň nejlepší, bylo 0,038. Při dalším snižování již program zcela přestal brát v potaz tvary jednotlivých oblastí a jako výsledek segmentace upřednostňoval hlavně interpolované pixely náležící hraně kruhu. Zároveň se také pod tuto hodnotu mnohonásobně prodloužila doba potřebná pro výpočet. Po dlouhých experimentech se zpracovávaným počtem vlastních vektorů a hodnotami σ jsem byl schopen nalézt takový výsledek, který se ideální segmentaci blíží nejvíce. Výsledek je zachycen na následujícím obrázku 28. Program byl i přes obtížnost vstupního problému nakonec schopen celkem obstojně lokalizovat jednotlivé oblasti popředí, které trpí pouze minimálním přetečením do nesprávných oblastí, což považuji za úspěch.

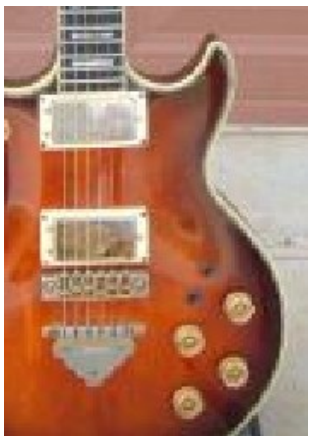


28 – Nejlepší dosažený segmentační výsledek čtvrtého umělého obrázku

6.1.2 Reálné obrázky

Prvním ze vstupních obrázků, které zachycují objekty reálného světa, jsem zvolil výřez fotografie, na které je elektrická kytara. Původně jsem na vstup mého programu nahrál obrázek celý, avšak po zjištění délky doby běhu výpočtu vlastních čísel a jejich vektorů jsem z něj raději vyřízl pouze nejzajímavější část. Na výřezu z celé fotografie je tudíž spodní část s tělem kytary, na které jsou obdélníkové snímače, část krku s pražci, čtyři kulaté knoflíky, struník a další drobné části. Hlavní oblastí by měl být samotný tvar těla kytary, který je navíc ohraničen bílou lemovkou. Na obrázku je

tedy pro segmentaci dostatek zajímavých oblastí. Aby se nejednalo o jednoduchý problém, vybral jsem navíc fotografii s různorodým pozadím a popředím s mírnými odlesky a změnami jasu. Vstupní obrázek je možno vidět na následující ukázce 29.



29 – První reálný vstup, výřez z barevné fotografie elektrické kytary

Obrázek dlouho odolával pokusům o úspěšné určení oblastí, které by se daly hodnotit jako přijatelné. Pro počáteční vyšší hodnoty parametru σ se tvary nalezených oblastí ani nepodobaly ničemu z obrázku. Po snížení hodnoty σ začaly být viditelné obrysy obdélníkových snímačů a postupným snižováním této hodnoty až na 0,045 jsem došel k výsledkům, se kterými již bylo možno pracovat. Určitou dobu mi také trvalo najít nejvhodnější počet vstupních vlastních vektorů pro finální shlukování (podrobněji rozebráno v samostatné kapitole), ale nakonec jsem byl schopen dojít k takovým výsledkům, které přijatelně znázorňují jednotlivé oblasti. Níže jsou k vidění tři nejzajímavější výsledky segmentace. Na levém výsledku program správně určil jednotlivé snímače jako samostatné oblasti a téměř se mu podařilo určit tvar těla kytary jako jeden celek. Krk se zde podařilo určit jako jeden segment, což se ve většině dalších případů nepodařilo vzhledem k pražcům, které svým podstatně rozdílným jasem toto značně znesnadňují. Prostřední a pravý obrázek dokladuje především snahu programu o správné určení tvaru těla kytary zároveň s nejistým určením oblasti krku s pražci. Nalezený počet kulatých knoflíků ve spodní části kytary se pro každé finální shlukování náhodně měnil, avšak najít správně všechny čtyři se mi za celou dobu nepodařilo ani jednou.

30 – Výsledky segmentace reálného obrázku s výřezem elektrické kytary

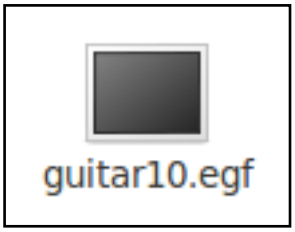
Jako další obrázek, který zachycuje reálné objekty, jsem zvolil výřez ze svojí portrétové fotografie. V praxi se dnes často můžeme setkat s aplikacemi, které pracují s detekcí tváře. Podobné algoritmy jsou implementovány i v kapesních fotoaparátech a mě tedy zajímalo, zda by můj program dokázal určit oblasti obličeje tak, aby pro něco podobného šly případně využít. Na výřezu z fotografie je tedy můj portrét s jasně viditelnými oblastmi tváře a pozadí, které není konstantní.

31 – Druhý reálný vstupní obrázek, zmenšená barevná portrétová fotografie

Jako v minulých případech, se zajímavé výsledky začaly objevovat až po snížení hodnoty σ na úroveň 0,05 a 0,04. Můj program byl schopen určit hlavní oblast popředí, kde zahrnul celý můj obličej včetně krku. Pro různé počty vstupních vlastních vektorů pro finální segmentaci se jen minimálně lišily další oblasti, které byl algoritmus schopen od sebe rozumně oddělit. Maximálním detailem tváře, který byl můj program schopen rozpoznat, byla oblast čela seshora ohraničená úrovní vlasů a zespod ohraničená tvarem obočí. Při nastavení vstupních parametrů programu pro nalezení těchto detailů však bylo pozadí rozbito na množství jednotlivých oblastí a výsledek tak vypadal velmi zmatečně. Žádný jiný jasný detail, který by byl využitelný pro rozpoznání obličeje, se programu ani po dlouhém experimentování nalézt nepodařilo. Na následující ukázce 32 jsou k vidění dva výstupy, kde program nejlépe určil tvar popředí a vysegmentoval správně mou hlavu společně s krkem. V jednom případě dokonce určil pozadí jako jeden celek. K lepšímu výsledku se mi s mým programem nepodařilo dojít. I přesto si však myslím, že tento výsledek není špatný.

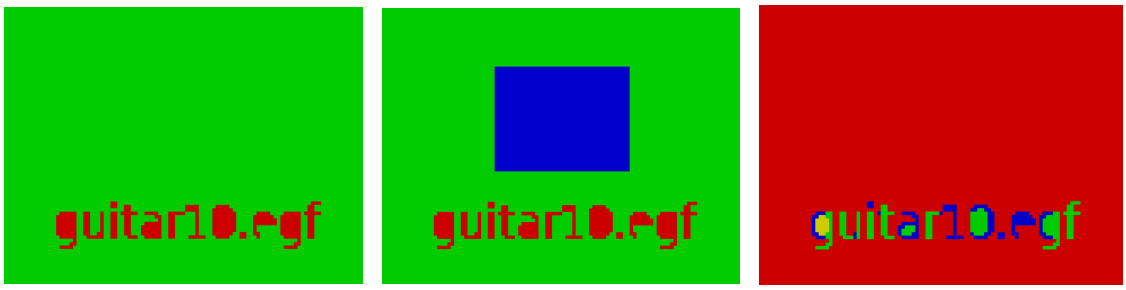
32 – Nejlepší dosažené výsledky segmentace zmenšené portrétové fotografie

Posledním vstupem, se kterým se v praxi můžeme setkat, jsem zvolil obrázek s obdélníkovým objektem a popisným textem. Algoritmy, které se zabývají nalezením a rozeznáním textu, nacházejí v praxi své uplatnění, o čemž jsem se přesvědčil již ve své bakalářské práci. Zajímalo mě tedy, jak si můj program povede v této oblasti. Na následujícím obrázku 33 je tedy vidět konkrétní podoba vstupu pro můj program. Jedná se o obrázek velikosti 106×82 pixelů, který byl jako výřez pořízen ze snímku plochy mého Unixového systému. Pro tuto ukázku jsem záměrně zvolil obrázek, který netrpí negativními vlivy na segmentaci. Podobný příklad, kde je navíc přítomen obrazový šum, totiž zkoumám v následující podkaptiole.



33 – Třetí reálný vstup, obrázek s obdélníkovým objektem a popisným textem

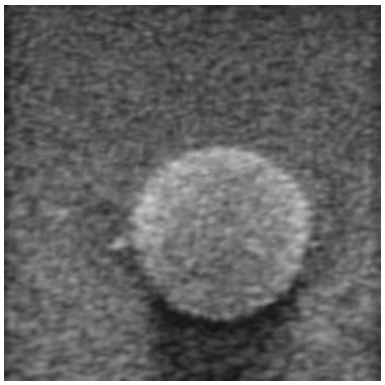
U tohoto obrázku se zajímavé a smysluplné výsledky začaly objevovat již při referenčním nastavení vstupních parametrů. Zanedlouho jsem tak po nikterak výrazném snížení hodnoty σ na 0,070 došel k pěkným výsledkům, na kterých bylo jasně zřetelné oddělení textu od obdélníkového okna a pozadí. Po krátkém experimentování s počtem vstupních vlastních vektorů jsem byl schopen obdržet výstupy, které jsou uvedeny níže. Na levém obrázku je text určen jako popředí a vše ostatní jako pozadí. V prostředním výsledku program určil navíc ještě i obdélníkové okno a správně jej označil jako další oblast. V posledním pravém výsledku je zobrazena snaha programu určit detaily písma za použití většího počtu vstupních vlastních vektorů do finální segmentace. Algoritmus zde byl schopen určit i vnitřky kulatých znaků, avšak nebyl již schopen správně slít takto rozdělený znak dohromady a nesprávně jej určil rozdělený na více oblastí. Pravý výsledek je asi nejlepší, který se mi pro tyto detaily podařilo z programu dostat.



34 – Výsledky segmentace obrázku s obdélníkovým objektem a popisným textem

6.1.3 Složité obrázky

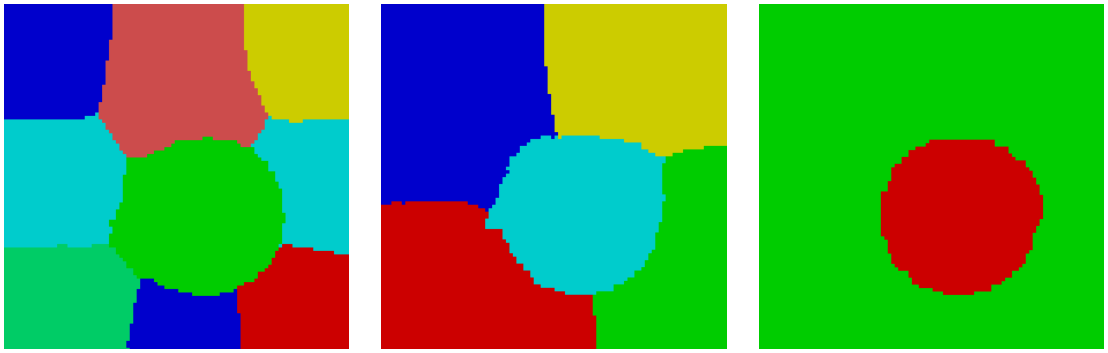
Prvním obrázkem, u kterého je správné určení oblastí velmi nesnadné, jsem zvolil snímek kruhové oblasti, která je značně zatížená šumem. Se snímky tohoto typu se můžeme často setkat například ve zdravotnictví při práci s výsledky různých vyšetření jako je rentgen či ultrazvuk. Na správném určení oblastí z takovýchto snímků tedy může záviset volba následné diagnózy, což znatelně přikládá této úloze na očekávané kvalitě výstupu. Vzhledem k tomu, že metoda, kterou můj program používá, není triviální, jsem byl zvědav, jak si s tímto problémem poradí. Na následující ukázce 35 je možno vidět konkrétní vstupní obrázek, který má velikost 220×220 pixelů.



35 – První těžce segmentovatelný obrázek, kruhová oblast zatížená značným šumem

Pro referenční hodnoty byl výpočet dokončen překvapivě rychle. Dle mého očekávání však jeho výsledek nepřinesl vůbec žádné informace o oblastech, které by bylo možno jakkoli využít. Snižováním hodnot parametru σ na úroveň okolo 0,05 jsem se však vcelku rychle dostal k výsledkům, které začaly naznačovat polohu námi hledaného kruhové objektu. Po snížení hodnoty parametru σ na 0,03 jsem byl schopen po menším experimentování s počtem vstupních vlastních vektorů správně určit celý objekt jako jeden celek zároveň se správným určením jednolitého pozadí. Na níže uvedených výsledcích je možno vidět toto postupné zlepšování kvality výsledku. Na levém obrázku již byla vidět snaha programu o nalezení kruhové oblasti. V prostředním případě již byl program velmi blízko správnému určení oblastí a zbývalo doladit detaily jemnou změnou vstupních parametrů a experimentováním. Na pravém obrázku již je vidět správně určený objekt našeho zájmu společně s jasně určeným pozadím. Zpracování tohoto obrázku mě velmi překvapilo tím, jak krátkou dobu trval výpočet vlastních čísel a jejich vektorů i přes to, že se nejedná o malý obrázek. Také kvalita finálního

určení objektu i přes značnou přítomnost šumu mě mile překvapila a dokládá tedy použitelnost této metody v praxi.



36 – Výsledky segmentace prvního těžkého obrázku

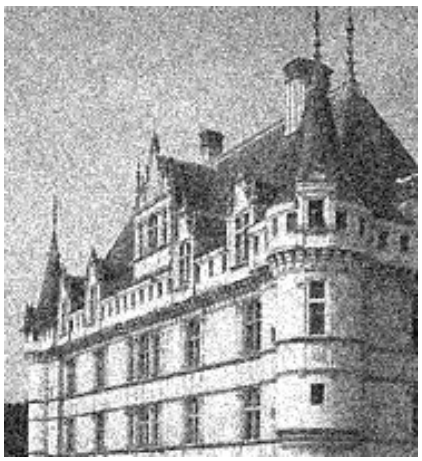
Jako další příklad špatně segmentovatelného obrázku jsem zvolil výřez ze známého obrázku jménem Lena. Našel jsem verzi, která byla zatížena šumem, a udělal z ní výřez o velikosti přibližně 170×170 pixelů, který obsahoval tvář ženy, která je na snímku vyobrazena. Po zadání tohoto výřezu na vstup při nastavení referenčních hodnot jsem čekal na výsledek vcelku dlouho a výsledek byl zcela nepoužitelný. Aby algoritmus určil nějaké relevantní oblasti, bylo mi jasné, že je nutno snížit hodnotu σ. Začal jsem tedy mírným snížením na 0,07 jako obvykle, abych viděl, jakým způsobem se výsledky zlepší. Na výsledek jsem však čekal více než 11 minut, načež jsem zjistil, že jsou rovněž nepoužitelné. Aby bylo možné z výsledků něco určit, musela by se hodnota σ snížit ještě více. Poté, co jsem vyzkoušel takto učinit, jsem po půlhodině bezvýsledného vytížení jádra mého procesoru na 100% výpočet ukončil. Výřez jsem pak zmenšil na velikost 135×45 pixelů a doufal, že se situace znatelně zlepší. Tento výřez je možno vidět na následujícím obrázku 37.

37 – Druhý težce segmentovatelný obrázek, výřez z portrétu zatíženého šumem

I pro tento malý výřez trval výpočet velmi dlouho. Pro hodnotu σ nastavenou na 0,06 jsem opět neobdržel od programu použitelné výsledky a snížil ji proto až na 0,04. Pro toto nastavení výpočet trval více než 15 minut a opět nepřinesl nic nového, co by se dalo použít pro finální shlukování metodou k-means. Pro tento vstup jsem tedy nebyl schopen vůbec určit jednotlivé oblasti ani pro velmi ořezaný úsek původního obrázku. Tento překvapivý závěr tedy jednoznačně říká, že metoda spektrálního shlukování je velmi závislá na volbě vstupního obrázku a rozhodně si neporadí se vším.

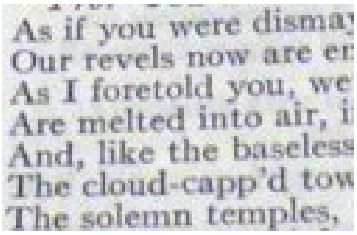
Úplně stejný výsledek měla má snaha o segmentaci zašumělého obrázku, na kterém je vyobrazen černobílý zámek. Z obrázku jsem udělal pouze výřez podobné velikosti jako původní výřez minulého příkladu, tedy přibližně 170×170 pixelů. Pro volbu vstupního parametru σ na 0,09, což vždy vedlo k rychlému a nekvalitnímu výsledku pro začátek, jsem byl nucen po půl hodině výpočet zastavit, neboť se program stále nedobral k žádnému výsledku a operační systém se díky delšímu přehřátí začal

již chovat velmi divně a přestával reagovat (teplota procesoru byla dlouhodobě nad vysokých 93°). Výřez vstupního obrázku, pro který můj program tedy rovněž nebyl schopen nic určit, je možno vidět níže.



38 – Třetí těžce segmentovatelný obrázek, výřez fotografie se zámkem zatížená šumem

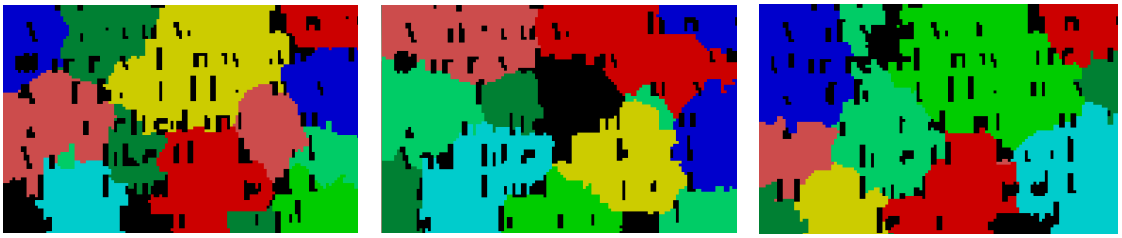
Vzhledem k příkladu s textem z předchozí kapitoly, jsem se rozhodl provést ještě jeden experiment, který by ověřil schopnost mého programu nalézt text na popředí z obrázku s horšími vlastnostmi. Většina skenovaných dokumentů, či vyfotografovaných textů trpí negativními vlivy, jako jsou ztráta kvality díky datové kompresi do ztrátových formátů, či šumem. Další vstup, který je vidět na dalším obrázku 39, tedy představoval výřez právě z takového textu. Díky kompresi jsou v něm hrany znaků rozostřeny a pozadí nemá jednotný jas.



39 – Čtvrtý těžce segmentovatelný obrázek, výřez naskenovaného textu zatížený šumem a komprimací

Zpočátku se pro vyšší hodnoty parametru σ opět nedařilo nalézt použitelné výsledky, které by umožňovaly finálním shlukováním určit jakoukoli užitečnou informaci. Při snižování hodnoty σ na úroveň kolem hodnoty 0,06 až 0,05 už program začal naznačovat polohu jednotlivých znaků, avšak výsledky byly velmi chaoticky uspořádány do jednotlivých oblastí. Následující tři snímky ukázky 40 zachycují zřejmě nejlepší výsledky, které mým programem šlo určit. Na všech uvedených snímcích je zřetelně vidět, jaký problém dělalo programu rušivé pozadí při hledání drobných detailů, kterými se řídí tvary jednotlivých znaků. Části znaků, které jsou vyobrazeny černě, se programu sice podařilo nějakým způsobem nalézt, avšak takovýto výsledek je pro další zpracování a rozpoznávání textu k ničemu. Nižší hodnotu parametru σ jsem ani nezkoušel zadávat, neboť výpočet již tak trval až 16

minut a jeho kvalita nějak výrazně nestoupala jako u jiných předchozích příkladů. Pro segmentaci textu tohoto charakteru se tedy dle mého metoda spektrálního shlukování příliš nehodí.

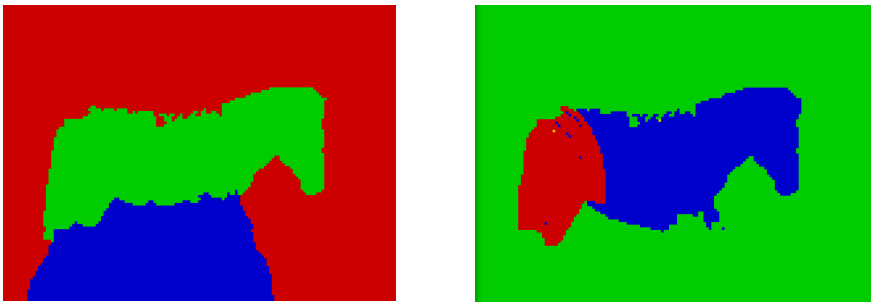


40 – Výsledky segmentace naskenovaného textu s šumem a komprimací

Posledním nelehkým vstupem pro můj program byla fotografie zebry. Toto zvíře jsme se za dobu mého studia pokoušeli segmentovat mnohokrát. Většina jednoduchých metod nebyla schopna určit zebru jako jednu oblast. Až komplexní metody, které jsme vyvíjeli dlouhé týdny, byly schopny si s tímto nějakým způsobem poradit. Byl jsem tedy zvědav, jaký výsledek bude mít použití mé metody spektrálního shlukování. Obrázek 41, který jsem tedy programu dal jako vstup tohoto problému, je uveden níže.

41 – Poslední těžce segmentovatelný obrázek, zmenšená barevná fotografie zebry

Výpočet jsem po předchozích zkušenostech začínal s nastavením parametru σ na 0,08. Výsledek, který jsem obdržel, toho mnoho nenaznačoval, ale určité náznaky tvaru zebry se již daly z náhodně roztroušených segmentů vydedukovat. Jelikož jsem se chtěl dostat k výsledku co nejdříve, zvolil jsem následně hodnotu σ rovnu 0,05. Výpočet trval bezmála půl hodiny a výsledkem byly pouze náhodné pixely na jednolité ploše pozadí. Tato hodnota již tedy byla příliš nízká. Pro hodnoty parametru σ kolem 0,059 jsem obdržel nejlepší výsledky, na základě kterých se podařilo většinu z tvaru zebry nalézt. Na následujících ukázkách výstupu mého programu jsou zachyceny dva nejlepší případy segmentace, které se mi podařilo dosáhnout. Nohy zebry se mi nepodařilo správně určit nikdy, zřejmě z důvodu neostrého přechodu mezi nimi a pozadím. Program však tuto úlohu zvládl daleko lépe, než bych očekával a většinu tvaru zebry s drobnými chybami nalézt zvládl, což mě pozitivně překvapilo.



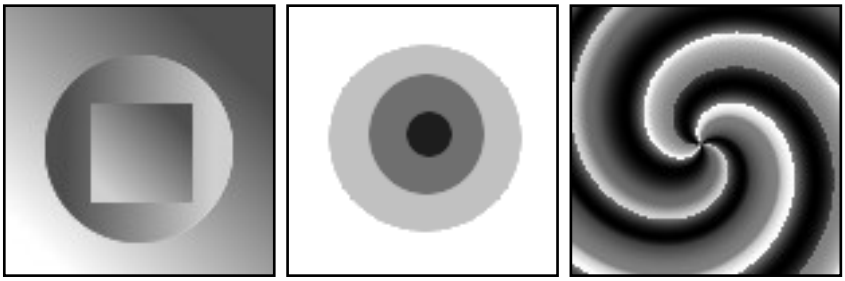
42 – Nejlepší dosažené výsledky segmentace obrázku se zebrou

6.2 Časová náročnost

Jak již bylo nastíněno v předchozích kapitolách, čas potřebný pro hlavní výpočet programu, kde se pomocí knihoven ARPACKu počítají vlastní čísla a jejich vektory vstupní Laplaceovy matice, se značně lišil v závislosti na více faktorech. Tyto závislosti se budu v následujících podkapitolách snažit určitým způsobem vysvětlit. Hlavní důraz však bude kladen na konkrétní výsledky, kterých jsem při práci s mým programem dosáhl při zpracovávání jednotlivých vstupních problémů. Program jsem spouštěl pouze na svém notebooku, který je vybaven procesorem Intel i5 U520 pracující na frekvenci až 1,8 GHz, a 4 GB operační paměti. Program jsem spouštěl mimo vývojové prostředí prostřednictvím systémové konzole v operačním systému Ubuntu, čímž jsem jeho běh urychlil. Mnohdy byly doby výpočtu velmi dlouhé a dosahovaly řádově až několika minut. Program totiž pro svůj běh používal pouze jednu čtvrtinu výkonu, kterou můj procesor nabízí, tedy jedno jádro, které je schopno zpracovávat naráz až dvě vlákna. Jedno ze čtyř logických jader, které bylo viditelné v systému, bylo tedy vždy po celou dobu výpočtu vytíženo na plných 100%. Jelikož se procesory řady i5 umí samy taktovat podle potřeby momentálního výkonu, mohl procesor pracovat v rozmezí frekvencí 1 GHz až 1,8 GHz. O této vrchní hranici však silně pochybuji, hlavně vzhledem ke stavu ovladačů pro Linux, kdy jsem tuto funkci nemohl nijak ověřit, či si ji v systému znázornit. Věřím, že při použití paralelní verze PARPACK, by dosahované časy byly znatelně kratší.

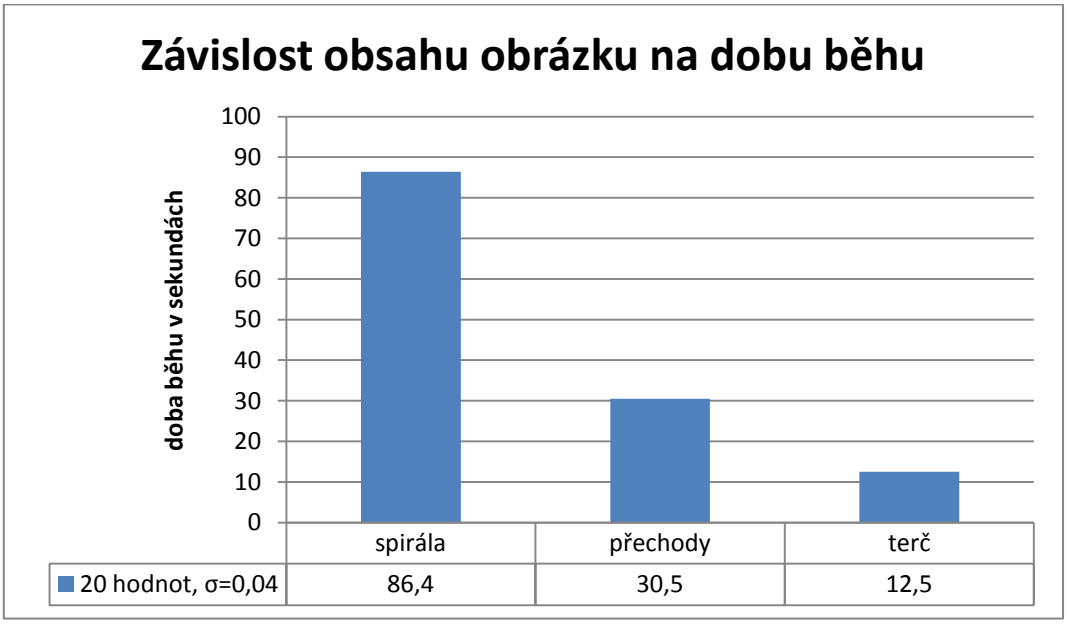
6.2.1 Vliv samotného obrázku

Jelikož se doby výpočtu lišily pro stejnou hodnotu vstupního parametru σ i pro stejné velikosti obrázků, došel jsem k závěru, že bude dobré se podívat i na vliv samotného obsahu obrázku. Pro potřeby porovnání vlivu obsahu obrázku jsem zvolil dva příklady, které se vyskytly v předchozích kapitolách společně s jedním, který jsem v textu neuváděl. Všechny tři mají shodnou velikost 100×100 pixelů a jsou k vidění na níže uvedené ukázce 43. Pro svou shodnou velikost budou tyto obrázky použity také v dalších podkapitolách sekce o vlivu na dobu běhu.



43 – Tři vstupní obrázky, na kterých budou popsány následující závisloti

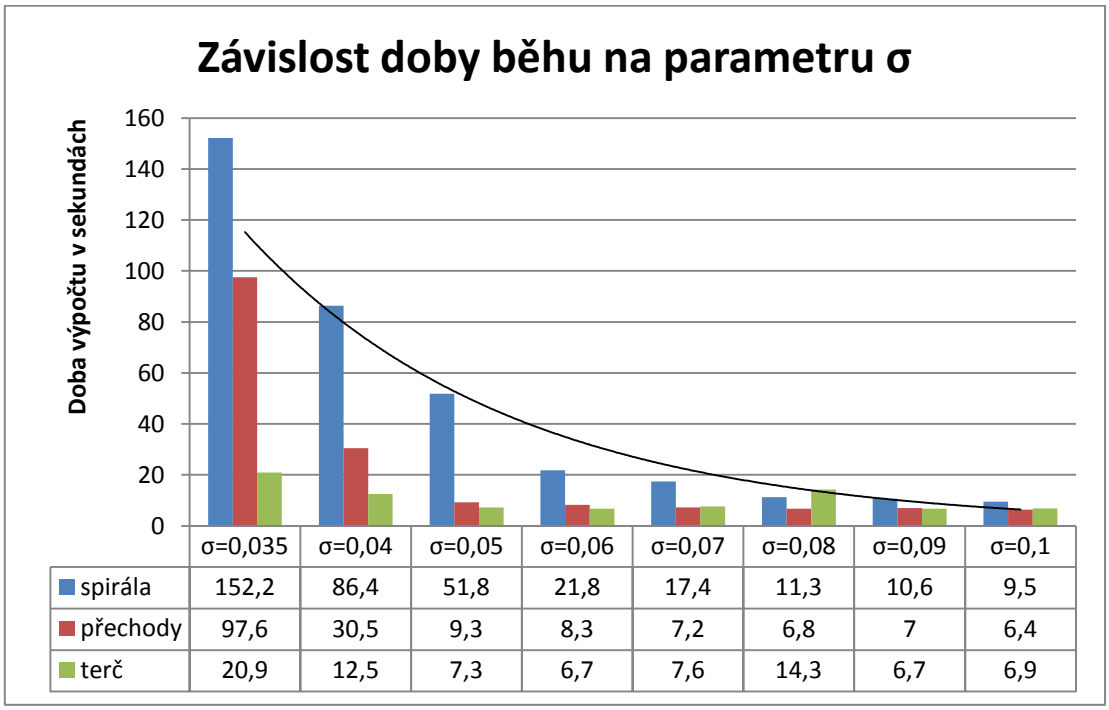
Na následujícím grafu 44 je srozumitelně znázorněno, jak dlouho trvaly jednotlivé výpočty pro tyto vstupní obrázky, které jsem nazval po řadě z leva jako přechody, terč a spirála. Výpočet jsem pro každý z nich spustil třikrát pro dané nastavení vstupních parametrů a dobu potřebnou pro vypočtení vlastních čísel a jejich vektorů zprůměroval. Při jednotlivých spuštěních pro stejné nastavení se doby běhu totiž lišily. Většinou se od sebe lišily maximálně okolo 5%, avšak v krajních případech se doba běhu různila až o 20%. Tento fakt mě však při zpracování na CPU nijak nepřekvapuje. Z grafu 44 je vidět, že ačkoli mají obrázky stejnou velikost, jejich zpracování trvá různě dlouho v závislosti na jejich obsahu. Všechny výpočty vrátily vždy 20 vlastních čísel společně s jejich vlastními vektory, což je počet, který jsem od programu nejčastěji požadoval pro své experimentování s dalším shlukováním pomocí k-means. Hodnota σ je zde nastavena na 0,04, což byla hodnota, která produkovala většinou jedny z nejlepších výsledků. Největší problém měl program s výpočtem hodnot pro spirálu, která má nejméně vyrovnaný histogram jasů svých pixelů. Následuje obrázek přechody, který má histogram obsazen víceméně rovnoměrně. Nejlépe je na tom terč, kde se kromě hran kružnic vyskytují pouze tři hodnoty jasu.



44 – Graf závislosti obsahu obrázku na dobu běhu výpočtu vlastních čísel a jejich vektorů

6.2.2 Vliv hodnoty σ

V této kapitole se podíváme na vliv hlavního vstupního parametru σ, který zásadně ovlivňuje kvalitu výsledků. Jako vstupy budou použity obrázky uvedené v předchozí kapitole. Počet požadovaných vlastních čísel a jejich vektorů bude také shodně nastaven na 20. Následující graf 45 zobrazuje přehledně jednotlivé doby běhu výpočtu vlastních čísel a jejich vektorů pro různá nastavení vstupního parametru σ. Situace ohledně složitosti zpracování jednotlivých obrázků je stejná jako v minulém příkladě, kdy nejvíce dalo programu zabrat vypočíst hodnoty pro obrázek se spirálou. K dobám výpočtu tohoto obrázku je v následujícím grafu 45 také přiměřena exponenciální funkce, kterou jak patrno doby běhu přibližně mají. Uvedené doby běhu byly opět zprůměrovány ze tří spuštění pro každé nastavení zvlášť. Z grafu 45 vyplývá, že s volbou hodnoty parametru σ musíme být opatrní zvláště pro větší vstupní obrázky. Při zadání velmi nízké hodnoty se totiž výpočet mnohonásobně prodlouží, a pokud zvolíme σ nižší než určitou hranici, pak výsledek navíc bude nepoužitelný. Tato hranice je závislá na obsahu obrázku a tudíž jí není možné znát dopředu. Vzhledem ke zkušenostem nabytým experimentováním s mým programem mohu doporučit začínat s hodnotou pro σ kolem 0,06 a 0,05, pokud chceme použitelné výsledky obdržet po co možná nejmenším počtu experimentů v rozumném čase.



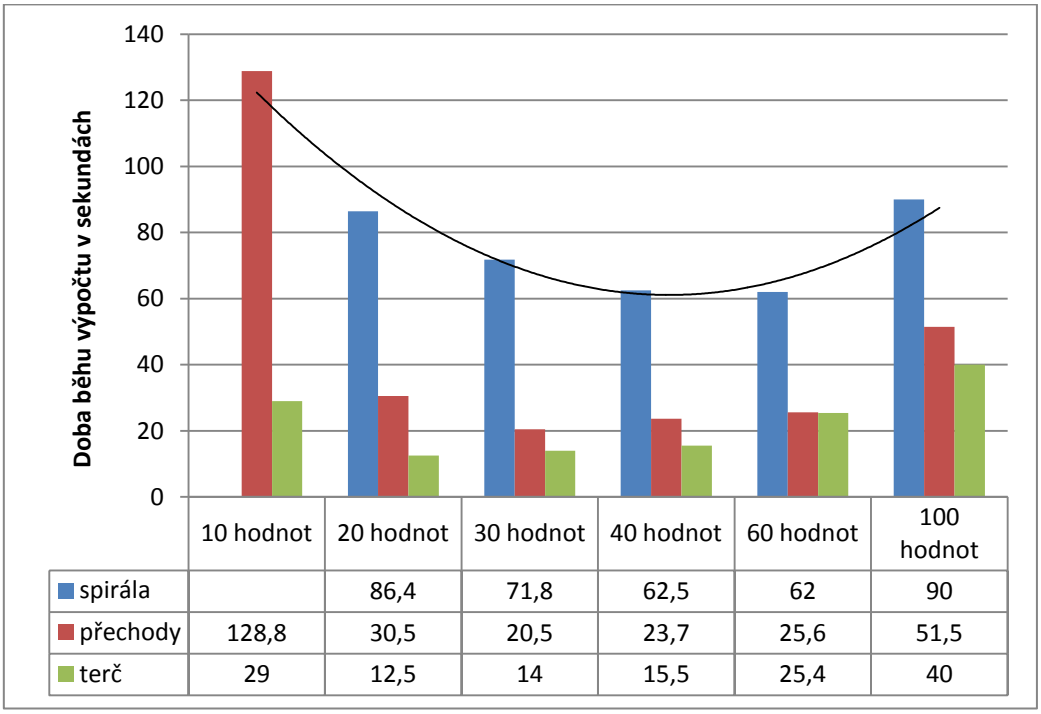
	σ=0,035	σ=0,04	σ=0,05	σ=0,06	σ=0,07	σ=0,08	σ=0,09	σ=0,1
spirála	152,2	86,4	51,8	21,8	17,4	11,3	10,6	9,5
přechody	97,6	30,5	9,3	8,3	7,2	6,8	7	6,4
terč	20,9	12,5	7,3	6,7	7,6	14,3	6,7	6,9

45 – Graf závislosti nastavení hodnoty parametru σ na dobu běhu výpočtu vlastních čísel a jejich vektorů

6.2.3 Vliv počtu požadovaných výsledků

Dobu běhu pro určité nastavení parametru σ a daný obrázek také ovlivňuje požadovaný počet vlastních čísel a jejich vektorů, který chceme od knihoven ARPACKu vypočíst. Na první pohled by člověka jistě napadlo, že tato závislost bude přímo úměrná k počtu požadovaných výsledků, a že čím více vlastních čísel a jejich vektorů budeme od knihoven požadovat, tím bude výpočet delší. Není tomu tak. Jak ukazuje následující graf 46, závislost doby běhu není ani lineární. Jako v předchozích

příkladech jsem opět použil již známé tři vstupní obrázky se stejnou velikostí a podrobil je výpočtům se stejným nastavením vstupního parametru σ. V grafu 46 jsou zachyceny průměrné časy výpočtu pro jednotlivé počty požadovaných vlastních čísel a jejich vektorů, které jsem opět průměroval ze tří spuštění pro každý z nich. Čas doby běhu pro obrázek spirály s nastavením deseti požadovaných hodnot jsem do grafu neuváděl, neboť jeho extrémně vysoká hodnota by graf znepřehlednila i za použití logaritmického měřítka. Vzhledem k vývoji hodnot je však trend dobře viditelný a závislost se snaží modelovat polynomická spojnice, kterou jsem do grafu nechal vypočíst. Z ní vyplývá, že nejkratší doba běhu byla vždy pro výpočet okolo padesáti hodnot, což se za celou dobu mého experimentování s programem mnohokrát potvrdilo. Pro finální určení oblastí pomocí některého z dostupných shlukovacích algoritmů však není dobré mnohdy zahrnovat do výpočtu všechny tyto vypočtené vektory. Ve svém programu toto řeším procedurou, která z uloženého souboru načte požadovaný počet vypočtených vlastních čísel a jejich vektorů, který si uživatel může zvolit vstupem z klávesnice.



	10 hodnot	20 hodnot	30 hodnot	40 hodnot	60 hodnot	100 hodnot
spirála		86,4	71,8	62,5	62	90
přechody	128,8	30,5	20,5	23,7	25,6	51,5
terč	29	12,5	14	15,5	25,4	40

46 – Graf závislosti požadovaného počtu vypočtených vlastních čísel a jejich vektorů na dobu nutnou pro jejich výpočet

6.3 Vliv vstupních parametrů na výsledky

Tato kapitola bude zaměřena na popis vlivu nastavení vstupních parametrů na kvalitu výsledků. Během své práce s programem, kdy jsem se snažil o co nejlepší segmentaci jednotlivých popisovaných obrázků, jsem získal určité zkušenosti s volbou hodnot vstupních parametrů. Pro jednotlivé obrázky navíc byly vstupní hodnoty, které produkovaly nejlepší výsledky, rozdílné. V následujících podkapitolách se tedy budu snažit objasnit, co jednotlivé vstupní parametry ovlivňují, a pro různá nastavení situaci ukážu na reálných výstupech z mého programu. Ke každému příkladu bude uveden komentář, který bude vysvětlovat podmínky a aktuální hodnoty. V programu je

nejdůležitější volba hodnoty pro požadovaný počet výsledných vlastních čísel a jejich vektorů a dále volba hodnoty σ. Tato volba správných hodnot vstupních parametrů není obecně v literatuře vysvětlena, a tudíž se mnohdy jedná o značně experimentální proces, který aplikuje metodu pokusu a omylu. Díky zkušenostem s výsledky této práce však možná bude pro další experimenty jednodušší pochopit, jak co nejlépe hodnoty zvolit již z počátku.

V literatuře jsou zmíněny referenční hodnoty, od kterých je dobré začít se zpracováním. Především se jedná o volbu vstupního parametru σ. Dle dostupných zdrojů je rozumné jeho hodnotu v počátku volit v rozmezí deseti až třiceti procent rozsahu jasu. V mé práci počítám s jasy, které reprezentuji hodnotami od nuly do jedné v datovém typu double. Jako referenční hodnotu jsem tedy volil nejnižší doporučenou, tedy σ=0,1. Počet požadovaných vlastních vektorů jsem pak volil v závislosti na vstupním obrázku, kdy se mi ze začátku experimentálně potvrdilo, že dobré výsledky lze obdržet například volbou takového počtu výsledných vlastních čísel a jejich vektorů, kolik oblastí chci v obrázku nalézt. Většina zdrojů se tomuto počtu nevěnovala a mnohé z nich navíc byly založeny na principu rekurzivní segmentace dle druhého nejmenšího vlastního vektoru.

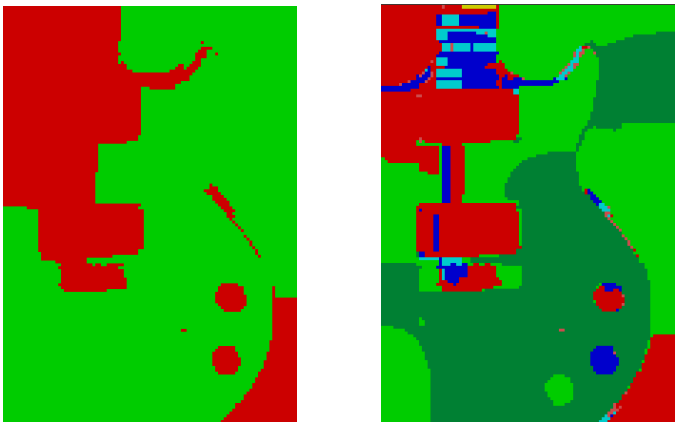
6.3.1 Vliv počtu požadovaných výsledků

Jako prvním vstupním argumentem pro hlavní výpočet vlastních čísel a jejich vektorů je pro program jejich požadovaný počet. Jak již bylo zmíněno v jedné z předchozích kapitol, tato hodnota znatelně ovlivňuje dobu nutnou pro nalezení těchto výsledků. V ideálním případě by bylo nejlepší programem vypočíst co největší množství vlastních čísel a jejich vektorů s co největší přesností a poté s nimi experimentovat, či využít pouze část z nich. Ne vždy však potřebujeme těchto vlastních vektorů mnoho. V případě, kdy by nám jich stačilo pouze méně, bychom počítač zbytečně zatěžovali a prodlužovali si tak čekání na výsledek. V následujících ukázkách tedy proberu vliv počtu vlastních vektorů, které budou předloženy na vstup metody finálního shlukování algoritmem k-means. Jelikož tento počet ovlivňuje výsledek vždy podobným způsobem, rozhodl jsem se jej hlavně demonstrovat na jednom větším příkladu za všechny.

Pro nejlepší popis tohoto vlivu jsem znovu vybral vstupní obrázek s částí kytary, který byl k vidění jako příklad v jedné z předchozích kapitol. Vybral jsem jej z toho důvodu, že obsahuje jak hlavní tvar popředí, tak větší počet menších detailů, které jsou s popředím svázány. Hlavní tvar zde reprezentuje samotné tělo kytary. Vedlejšími detaily jsou pak kulaté knoflíky, struník, krk s pražci, či jednotlivé obdélníkové snímače. Všechny následující výsledky byly zpracovány mým programem pro vstupní hodnotu parametru σ rovnu 0,040. Následující ukázka znázorňuje situaci, kdy byly použity pouze dva vlastní vektory odpovídající dvěma nejmenším vlastním číslům. První z nich, který odpovídá vlastnímu číslu rovnu nule, v sobě neobsahuje žádnou informaci pro segmentaci, neboť má všechny hodnoty stejné. Druhý vektor však obsahuje zásadní informace, díky kterým je možno provést optimální řez vstupním grafem, tedy v našem případě optimálně rozdělit vstupní body na dva hlavní segmenty.

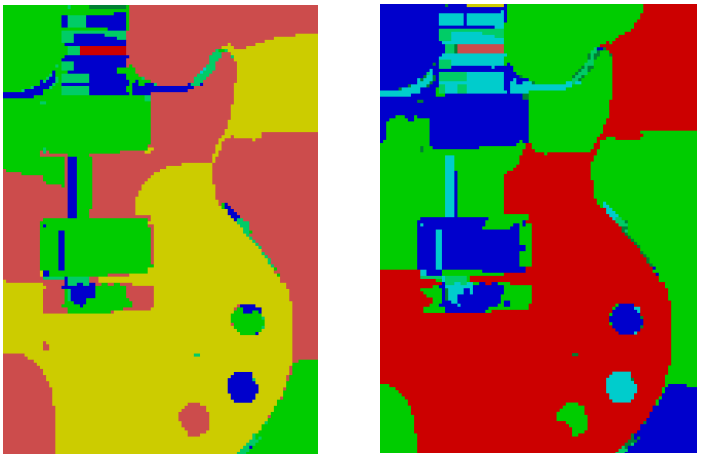
Tato situace je zachycena na levém snímku ukázky 47, kde jsou vidět hlavní rysy vstupního obrázku. Z obrázku lze snadno poznat tvar kytary, umístění obdélníkových snímačů, struníku, i dokonce dvou kulatých knoflíků. Člověk by z tohoto výsledku kytaru nejspíše poznal. Pro další zpracování je však

určitě nutno nalézt více samostatných jednotlivých oblastí. Na pravém snímku je ukázána situace, kdy jsem finální shlukovací metodu k-means nechal najít pro dva vstupní vlastní vektory osm oblastí. Situace je lepší než v obrázku nalevo, avšak určené oblasti jsou značně zmatečné. Detaily se prolínají s jinými cizími segmenty, ostré hrany přetékají do oblasti pozadí a menší oblasti jsou rozbity na více částí. Informace obsažená prakticky v jediném relevantním vektoru tedy ke správnému určení takových oblastí nestačí.



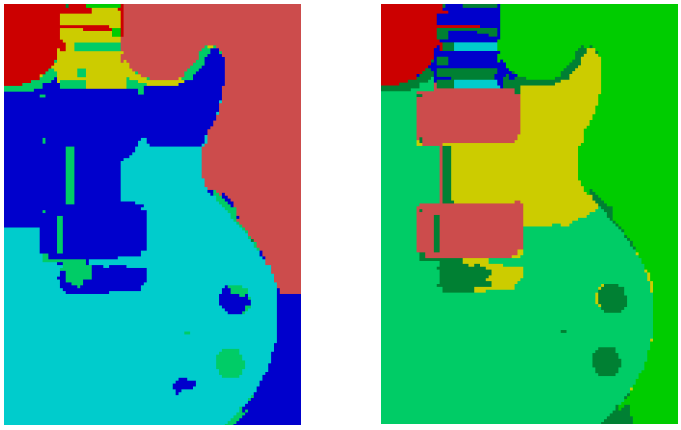
47 – Ukázka segmentace pro dva vlastní vektory při hledání dvou (vlevo) a osmi (vpravo) oblastí

Je vidět, že se počet vstupních vlastních vektorů musí zvýšit. V další ukázce 48 jsem nechal finální shlukovací metodu zpracovávat pět a deset vlastních vektorů. Požadovaný počet nalezených oblastí jsem nastavil opět na osm jako v minulém případě. Levý obrázek ukazuje situaci pro pět vstupních vektorů a pravý pro deset. Určení detailů a nových oblastí je stejně špatné jako v minulém případě, ne-li horší. Uvedené výsledky jsou nejlepší a nejrozumnější, jaké se mi podařilo segmentovat metodou s náhodným určením počátků. Výstupy se značně lišily pro každé spuštění a většina z nich bylo více zmatečných než tyto.



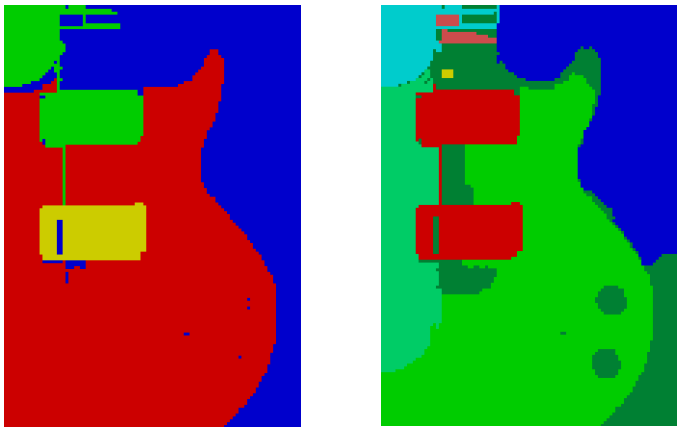
48 – Segmentace při zpracování pěti (vlevo) a deseti (vpravo) vlastních vektorů při hledání osmi oblastí

Následující ukázka 49 ukazuje situaci pro zpracování dvaceti a pětatřiceti vlastních vektorů, kdy program hledal opět osm oblastí. Levý obrázek odpovídá dvaceti vstupním vektorům pro finální shlukování. Zde se výsledek již určitým způsobem zlepšil a zmatečné oblasti se začaly více slévat do rozumných celků. Pravý obrázek s pětatřiceti vstupními vektory znázorňuje další postupné zlepšení. Zde už bylo daleko lépe určeno pozadí, jednotlivé obdélníkové snímače i oblast krku kytary. Detaily jako kulaté knoflíky už rovněž nejsou tak znepřehledněny cizími oblastmi, které by do nich zasahovaly jako v minulých případech.



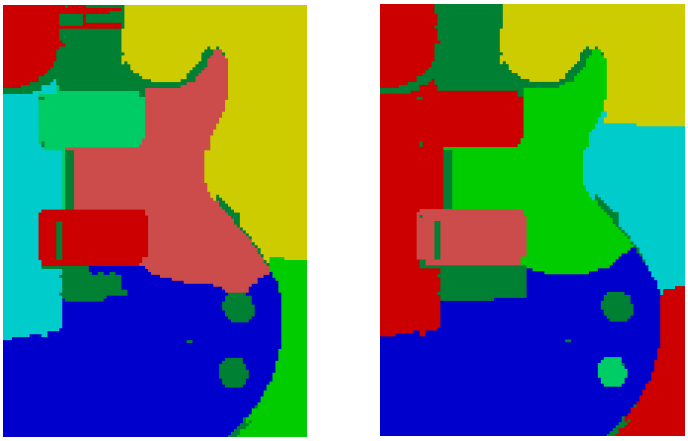
49 – Segmentace pro dvacet (vlevo) a pětatřicet (vpravo) vlastních vektorů při hledání osmi oblastí

Na další ukázce 50 je možno vidět výsledky pro použití padesáti vlastních vektorů. Pro situaci na levém obrázku jsem finální shlukování nechal nalézt pouze čtyři oblasti, abych ukázal nejlepší výsledek segmentace hlavního tvaru popředí, tedy tvaru těla kytary. Doposud, s nižším počtem vstupních vlastních vektorů, tohoto nebylo možno dosáhnout. Na levém obrázku je tak vidět nejlepší výsledek, kdy bylo co nejlépe určeno popředí jako jedna oblast společně s hlavními detaily, které tvoří obdélníkové snímače. Na pravém obrázku program hledal opět osm oblastí. Na rozdíl od předešlých příkladů zde již nebyl takový problém dospět k výsledkům, které neobsahovaly přetékání oblastí přes hrany do cizích segmentů. Většina výsledků dosažených metodou náhodného určení počátků měla správně oddělené pozadí od popředí, ačkoli tyto byly obvykle rozděleny na více samostatných oblastí. Výsledky však byly viditelně kvalitnější a srozumitelnější.



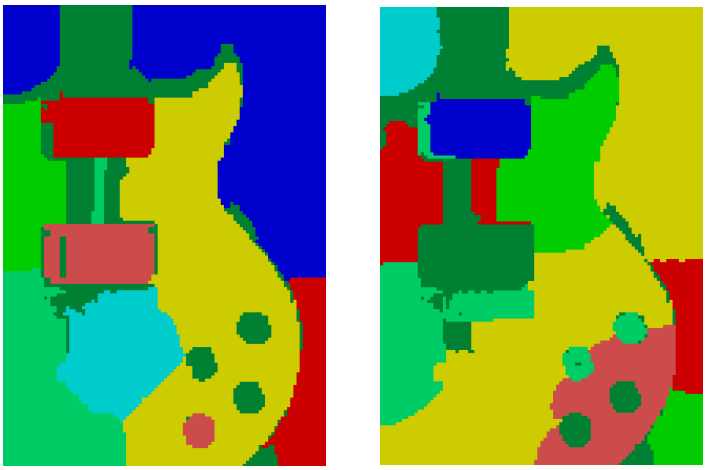
50 – Segmentace pro padesát vlastních vektorů při hledání čtyř (vlevo) a osmi (vpravo) oblastí

V předposlední ukázce 51 pro vstupní obrázek s výřezem kytary je možno vidět nejlepší výsledky pro volbu pětasedmdesáti a sta vlastních vektorů, kdy bylo požadováno nalezení osmi oblastí. Obrázek nalevo měl jako vstup pro finální shlukování pětasedmdesát vlastních vektorů. V drtivé většině případů program správně oddělil segmenty popředí od pozadí. Téměř nikdy se také nestalo, že by se nesprávné oblasti slévaly dohromady. Nikdy se však již pro tento počet (i vyšší) nepodařilo určit popředí jako jeden celek. Tvar těla kytary byl vždy rozbit na více jednotlivých segmentů. Pravý obrázek je výsledkem segmentace pro sto vstupních vektorů. Zde se již drobné detaily začaly velmi slévat do větších oblastí a málokdy tak například bylo možno určit oblasti jednotlivých pražců krku kytary. Také nalezení alespoň dvou kulatých knoflíků již nebylo tak časté jako v minulých případech.



51 – Segmentace pro pětasedmdesát (vlevo) a sto (vpravo) vlastních vektorů při hledání osmi oblastí

Posledními výsledky, které jsem nechal svému programu zpracovat, jsou snímky ukázky 52, které znázorňují použití sto padesáti a dvě sta vlastních vektorů pro finální shlukování pomocí k-means. Levý obrázek dokladuje použití sto padesáti vlastních vektorů, kdy se programu úspěšně dařilo téměř vždy správně najít všechny čtyři kulaté knoflíky, což předtím nebývalo běžné. I obdélníkové snímače jsou nalezeny téměř přesně, ale častěji bývají nesprávně spojeny s cizími oblastmi. Popředí se jako jeden celek nepodařilo nalézt nikdy. Pravý obrázek tedy ukazuje zřejmě nejlepší výsledek, který jsem byl schopen pro dvě stovky vstupních vlastních vektorů obdržet. Detaily pražců a krku kytary se téměř vždy slévaly dohromady a popředí kytary bylo vždy rozbito na více jednotlivých oblastí. Jelikož se oblasti volbou vyšších počtů vstupních vektorů začínají více rozpadat, algoritmus jsem pro větší počty netestoval. Kvalita výsledků totiž dle mého mínění poté již pro tento obrázek pouze klesá, což je možno pozorovat i na následujících dvou příkladech.



52 – Segmentace pro sto padesát (vlevo) a dvě stě (vpravo) vlastních vektorů při hledání osmi oblastí

6.3.2 Vliv hodnoty σ

V této kapitole se budu věnovat popisu vlivu nastavení hodnoty vstupního parametru σ. Správná volba hodnoty tohoto parametru je klíčová ke správné segmentaci vstupního obrázku a největší měrou ovlivňuje kvalitu výsledku. Hodnota, pro kterou program nalezne nejlepší výsledky, není stejná pro všechny obrázky. Její hodnota je závislá na obsahu daného vstupního obrazu. To, jakým způsobem, se budu snažit v této kapitole nastínit na názorných ukázkách, které jsem pořídil ze svého programu. Většinou jsem k nejvhodnější hodnotě tohoto parametru došel jejím postupným snižováním od referenční hodnoty 0,1. Pokud však hodnotu snížíme pod určitou hranici, přestanou být výsledky k jakémukoli užitku. I tato hranice byla závislá na vstupním obrázku.

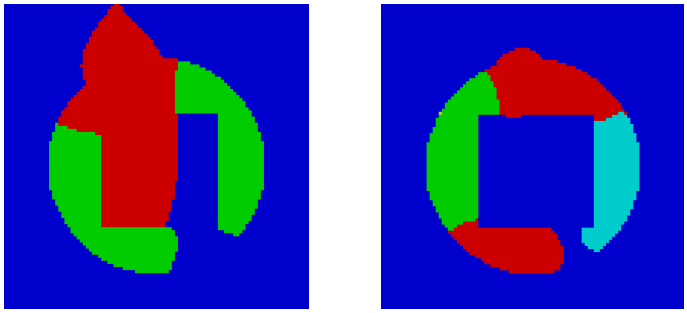
Rád bych také zmínil, že ačkoli hodnota parametru σ nejvíce ovlivňuje kvalitu výsledků, tak zároveň nejvíce prodlužuje dobu nutnou pro vypočtení vlastních čísel a jejich vektorů. Je tedy na uživateli programu, aby si rozmyslel, do jaké míry se bude snažit snižovat tuto hodnotu a tím déle čekat na výsledek. Kvalita výsledku se totiž také v závěru, kdy dáváme vypočtené vlastní vektory na vstup metodě k-means, dá vylepšit správnou volbou počtu těchto použitých vlastních vektorů. Ke správné segmentaci tak můžeme dojít buď tak, že budeme pro konstantní počet zpracovávaných vlastních vektorů zkoumat výsledky při snižování hodnoty parametru σ, nebo že zvolíme vyšší hodnotu parametru σ a dále budeme experimentovat s počtem zpracovávaných vlastních vektorů ve finálním shlukování. Metoda k-means, kterou jsem ve svém programu implementoval, totiž běží velmi svižně a je proto snadné rychle dojít k výsledkům, u kterých již vizuálně poznáme jejich kvalitu. Při použití metody náhodného určení počátků navíc můžeme výslednou segmentaci nechat určovat znovu a znovu a vybrat si výsledek, který se nám hodí nejvíce, což je mnohem rychlejší, než zadávat nový výpočet se změněným parametrem σ.

Jako hlavní příklad, na kterém budu demonstrovat vliv hodnoty vstupního parametru σ, jsem vybral již výše uvedený umělý obrázek 27 s jasovými přechody. Na tomto obrázku je totiž dle mého tento vliv nejvíce patrný. Manipulací s hodnotou σ pro segmentaci tohoto obrázku se řídilo především to, jak si program poradí se splývajícími hranami oblastí, na kterých došlo k přetékání objevených segmentů do nesprávné oblasti.

Na následujících třech ukázkových výstupech 53 jsou zachyceny nejlepší výsledky pro hodnoty σ z leva o hodnotách 0,1, 0,08 a 0,07. Levý obrázek pro referenční hodnotu dokladuje, jak zmatečně byly výsledné oblasti určeny, segmenty se slévají v libovolných kombinacích objektů pozadí o popředí. U prostředního obrázku je již výsledek o trochu lepší, stále však dochází k přetečení na spojených hranách se stejným jasem sousedních pixelů a rozbití hlavní kruhové oblasti na více segmentů. Pravý obrázek je tom velmi podobně. Tyto „vysoké“ hodnoty parametru σ tedy nejsou pro tento obrázek, kde hrany hrají hlavní roli, vhodné.

53 – Segmentace pro vyšší hodnoty parametru σ

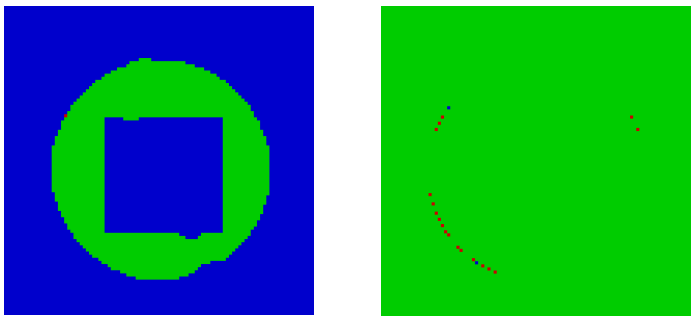
Následující dva snímky ukázky 54 se týkají výsledků pro hodnotu parametru σ z leva 0,06 a 0,05. Na levém obrázku je již patrné znatelné zlepšení výsledné segmentace, kdy se programu začalo dařit nalézat lépe hlavní tvar kruhového popředí s menšími oblastmi, které by přetékaly do cizích segmentů. Na pravém obrázku je situace ještě o něco lepší. Kruhová oblast je již skoro celá určena a čtvercový vnitřek byl pouze s malým přetečením určen také téměř správně. Ačkoli jsem provedl mnoho experimentů s finálním shlukováním pro tuto hodnotu, lepších výsledků se mi pro tuto hodnotu vstupního parametru σ dosáhnout nepodařilo.



54 – Segmentace pro nižší hodnoty parametru σ

Poslední ukázkou 55 pro tento vstupní obrázek jsou příklady výstupu, které jsem obdržel pro hodnoty parametru σ z leva rovny 0,04 a 0,03. Ačkoli není situace na levém obrázku ideální, jedná se o nejlepší určení oblastí, jakého bylo pro tento vstup možno dosáhnout, a to hlavně z důvodu nejmenšího přetečení vnitřní čtvercové oblasti směrem do vnějších částí. Program také nejlépe určil hlavní kruhovou oblast a pozadí jako jeden celek. Chybou však je, že vnitřní čtvercový segment patří dle programu k pozadí. Nalézt oblasti s touto kvalitou na tři jednotlivé správné segmenty se mi

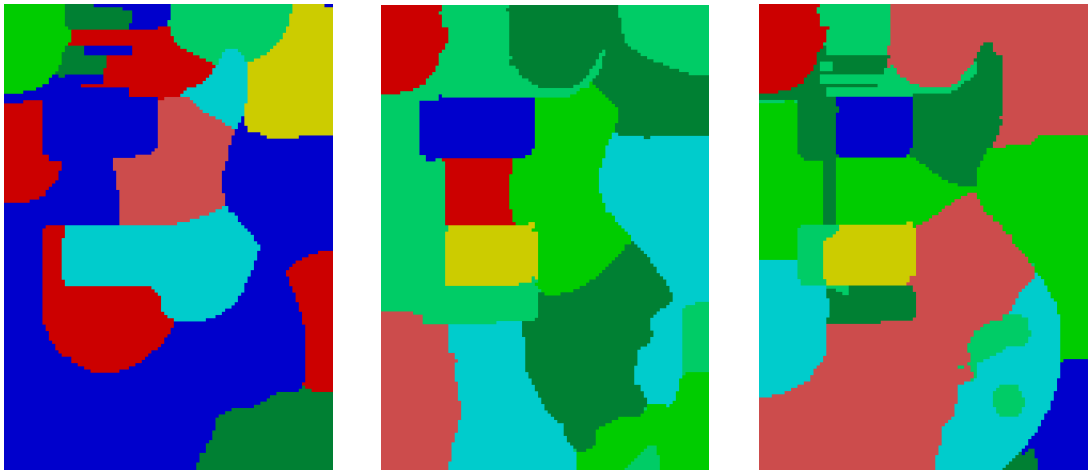
nepodařilo. Jelikož se však jednalo o vcelku náročný vstup, považuji tento výsledek za přijatelný. Na pravém obrázku je však situace pro hodnotu σ rovnu 0,03. Výsledná segmentace vypadá vždy stejně a zahrnuje pouze jednotlivé přechodové pixely vnější kruhové oblasti. Takovýto výsledek je zcela k ničemu, ačkoli jeho výpočet trval nejdéle. Pokud pro tento vstupní obrázek snížíme hodnotu parametru σ pod hodnotu 0,0380, začnou se výsledky rapidně zhoršovat a blížit tomuto výsledku.



55 – Segmentace pro nejnižší hodnoty parametru σ

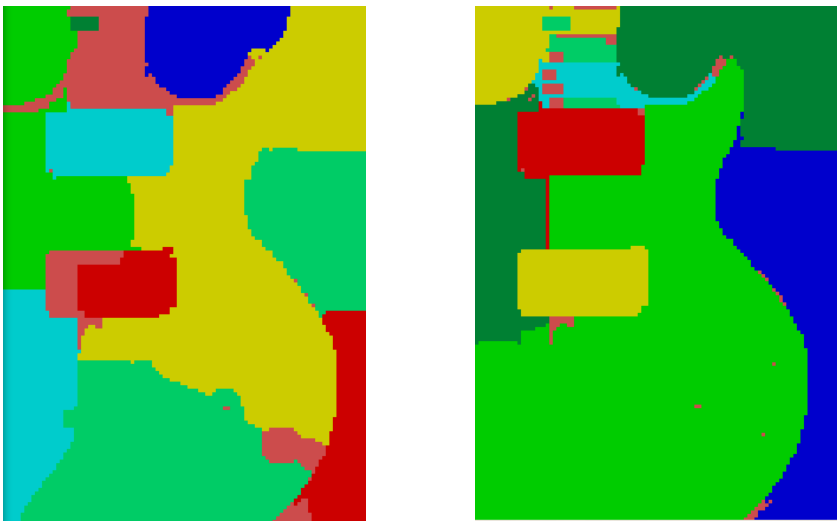
Jako další příklad jsem zvolil již známý obrázek 29 s výřezem elektrické kytary. Zde byl vliv nastavení parametru σ také dobře viditelný. Manipulace s jeho hodnotou především měnila váhu hran, které byly v obrázku s kytarou klíčové, neboť přes ně mnohdy také oblasti přetékaly do cizích segmentů, nejvíce do pozadí. Dále pak snižováním hodnoty parametru σ docházelo k daleko lepšímu nalezení detailů v obrázku, jako jsou kulaté knoflíky či správné určení obdélníkových snímačů bez rušení jejich rámečky či strunami.

Na následujících snímcích ukázky 56 je tedy z levé strany k vidění situace pro nastavení parametru σ pro referenční hodnotu 0,1, 0,08 a 0,06. Pro tyto hodnoty se výsledky svou kvalitou příliš neliší. Tak jako v minulém případě s obrázkem s přechody jsou určené oblasti zmatečné a rozbité na více samostatných segmentů, které mnohdy přetékají tam, kde nemají. Určité zlepšení je směrem k nižším hodnotám viditelné, avšak stále nedostačující. V pravém obrázku je již vidět snaha o korektnější nalezení detailů kulatých knoflíků a správné určení oblasti krku kytary.



56 – Segmentace pro vyšší hodnoty parametru σ, příklad druhý

Poslední dva snímky ukázky 57 znázorňují výsledky pro hodnoty parametru σ z leva 0,05 a 0,04. Na levém obrázku je situace lepší než na minulých příkladech. Horní obdélníkový snímač se lépe určen a obecně byl program při více pokusech lépe schopen správně určit kvalitnější segmenty. Na pravém obrázku je situace už velmi dobrá, což dokládá velmi pěkně určený tvar popředí bez jakéhokoli přetečení do oblasti pozadí. Opět jsem tedy dostal velice pěkný výsledek pro hodnotu σ rovnu 0,04. Na hranách těla kytary jsou však již vidět drobné rušivé pixely, které získávají snižováním hodnoty σ na váze, jak bylo ukázáno na předešlém příkladu s přechody. Výpočet pro hodnotu σ rovnu 0,04 zde trval mnoho minut a jeho výsledky jsou kvalitní. Vzhledem k rapidnímu růstu doby potřebné pro výpočet jsem dále tuto hodnotu nesnižoval. Myslím si navíc, hlavně vzhledem k minulému příkladu s přechody, že by výsledky o moc kvalitnější nebyly a mohlo by velmi brzy dojít k jejich znatelnému zhoršení. Můj systém navíc plné zatížení přes půl hodiny snášel velmi špatně, díky dlouhodobému přehřátí CPU.



57 – Segmentace pro nízké hodnoty parametru σ, příklad druhý

6.4 Shrnutí poznatků

Při testování algoritmu spektrálního shlukování pomocí mého programu, kde jsem tuto metodu implementoval, jsem došel k závěru, že tato metoda má své využití. Během experimentování s rozličnými vstupy jsem se dozvěděl, jaké druhy obrázků dělají této metodě problém, a které naopak zvládá velmi dobře. Uvědomil jsem si, že hlavní výpočet vlastních čísel a jejich vlastních vektorů je základem této metody, na kterém toho příliš změnit nelze.

Mnou popsané experimenty s volbou počtu zpracovávaných vlastních vektorů ukazují závislost výsledku na tomto počtu. Zde situace není až tak složitá. Pokud máme obrázek s množstvím detailů, je lepší finálním shlukováním zpracovávat větší počet vlastních vektorů. Čím více jich dáme na vstup finálního shlukování, tím více informací o menších samostatných oblastech nám poskytnou. Pokud se větší oblasti již začnou rozpadávat na jednotlivé menší segmenty, je lepší počet zpracovávaných vlastních vektorů mírně snížit. Pokud máme obrázek s oblastmi spíše bez detailů, tvořených převážně plochami bez rušivých pixelů, hran či šumu, je lepší volit vlastních vektorů méně. Obecně je tedy lepší

si vlastních čísel a jejich vlastních vektorů nechat programem vypočíst více, uložit je, a poté z nich načíst pouze určitý počet, abychom mnohdy zdlouhavý proces jejich výpočtu nemuseli spouštět znovu.

Podobně je tomu s volbou hodnoty parametru σ. Zde bych namísto referenční hodnoty 0,1 doporučil začínat spíše s hodnotou 0,06 pro jednodušší obrázky bez jemných detailů či šumu. Pro složitější obrázky, u kterých je nalezení detailů podstatné, bych pak počáteční hodnotu volil 0,05. Pokud v prvotních výsledcích pro tyto hodnoty bude málo detailů, či budou-li hlavní oblasti popředí stále mít tendenci přetékat přes hlavní hrany do cizích segmentů, hodnotu je třeba mírně snížit. Pod hranici 0,05 bych doporučoval hodnotu snižovat v krocích po 0,005. Jakmile se začnou výsledky znatelně zlepšovat, je lepší tuto hodnotu snižovat ještě jemněji. Kolem hodnot pod 0,04 se totiž mohou výsledky začít také rapidně zhoršovat, jak bylo ukázáno v jedné z předchozích kapitol.

Z experimentů jsem také získal jasný dojem, že výsledky mého programu a této metody obecně by mohly být znatelně lepší, pokud by se podařilo implementovat komplexnější algoritmus finálního shlukování. K výsledkům, které jsem ve své práci prezentoval jako obrázky ze svého programu, jsem totiž téměř vždy došel použitím metody k-means s náhodným určením počátků, ačkoli se mi z počátku zdála méně vhodná. Z toho mi vyplývá, že informace o oblastech, kterou výpočet vlastních čísel a jejich vektorů k původní reprezentaci vstupu přidává, se ve vypočtených vlastních vektorech nachází, avšak můj algoritmus ji neumí plně využít ke zlepšení kvality výstupní segmentace.

Algoritmus si dobře poradil se vstupními obrázky, které byly uměle vytvořeny a v nichž byly oblasti vcelku jasně odděleny. Problémy taky nedělaly ani složitější tvary takových oblastí, které by mohly většinu jednodušších segmentačních algoritmů zmást k nesprávnému určení segmentů. Metoda spektrálního shlukování také dle mého dobře fungovala na reálných obrázcích, které netrpěly negativními vlivy jako šum či příliš rušivé pozadí. Pro obrázky s větší přítomností šumu tato metoda dle mého vhodná není, pokud jsou na obrázcích složitější oblasti než jednoduché geometrické tvary, jak dokazuje jeden z mých experimentů.

7 Závěr

Úkolem této práce bylo experimentálně ověřit metodu spektrálního shlukování pro využití segmentace obrazu. Po nastudování problematiky týkající se jednoho z používaných algoritmů realizující tuto metodu jsem postupně navrhl a implementoval program, který jej využívá k segmentaci obrazu. Pro práci s obrazem jsem využil existující knihovny OpenCV, se kterými jsem se seznámil v rámci svého studia. Jelikož je správný výpočet vlastních čísel a jim odpovídajících vlastních vektorů velice komplexní záležitostí, rozhodl jsem se použít balík knihoven ARPACK, který byl za tímto účelem vyvinut. Po nesnadném zprovoznění těchto knihoven jsem sám implementoval finální shlukovací algoritmus k-means, který z vypočtených a transformovaných vlastních vektorů určí jednotlivé výsledné oblasti, které se nachází ve vstupním obraze.

Během vývoje programu jsem pozornost věnoval i jeho snadnému použití. Výpočet vlastních čísel a jejich vektorů zabírá mnoho výpočetního času, což jsem vyřešil jejich ukládáním do souboru pro pozdější zpracování, kdy je možno ručně určit počet použitých vlastních vektorů. Také u metody finálního shlukování algoritmem k-means je možno zvolit dvě metody stanovení jejich počátečních souřadnic. Tímto způsobem bylo možno zpracovat potřebné experimenty, které jsem potřeboval v této práci provést.

Na základě četných experimentů, které jsem provedl pomocí svého programu, jsem popsal mnoho závislostí a výsledků metody spektrálního shlukování. Právě tento úkol byl v mé práci tím podstatným a proto je mu věnována patřičná pozornost. Věřím, že výsledky, kterých se mi podařilo dosáhnout, jsou dobré a především přínosné pro případné další navazující práce v rámci této problematiky. Vzhledem k tomu, že v literatuře, se kterou jsem se o tomto tématu setkal, jsou mnou popsané závislosti probrány jen minimálně, věřím také, že jsou mé poznatky a závěry o to cennější.

Ačkoli jsem měl na celý vývoj, ladění a testování této práce omezené časové možnosti díky svému celoročnímu zahraničnímu pobytu, myslím, že zadání se mi podařilo splnit velmi dobře. Program neobsahuje závažné chyby a je připraven jak pro přímé používání, tak pro případné vylepšování, které jsem ve své práci rovněž navrhl a popsal.

Velkým přínosem pro mě bylo seznámení se s novým a komplexním balíčkem funkcí ARPACK a také zlepšení se v používání knihoven OpenCV. Také jsem se seznámil s pro mě zcela novým způsobem přístupu k segmentaci obrazu, což jistě rozšířilo mé obzory. Experimentování s programem mě pak naučilo soustředit se na to, co je při práci důležité, a proto jsem svůj program upravil pro snadnější použití.

8 Bibliografie

[1] Andrew Y. NG, M. I. (2002). *On Spectral Clustering: Analysis and Algorithm.* Získáno 27. 6 2012, z Carnegie Mellon University, CMU: http://www-2.cs.cmu.edu/Groups/NIPS/NIPS2001/papers/psgz/AA35.ps.gz

[2] *ARPACK Software*. (nedatováno). Získáno 11. 6 2012, z Computational & Applied Mathematics, Rice University: http://www.caam.rice.edu/software/ARPACK/

[3] Donath, W. E. (1973). *Lower Bounds for the Partitioning of Graphs.* Získáno 27. 6 2012, z IEEEXplore Digital Library: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5391366&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5391366

[4] Francis R. Bach, M. I. (2004). Learning Spectral Clustering. *Advances in Neural Information Processing Systems 16* (stránky 305-312). Massachusetts: Massachusetts Institute of Technology.

[5] Charles J. Alpert, A. B.-Z. (1999). *Spectral partitioning with multiple eigenvectors.* Získáno 27. 6 2012, z ScienceDirect.com: http://www.sciencedirect.com/science/journal/0166218X/90/1

[6] Chung, F. R. (1994). *Spectral Graph Theory, vydání 92.* Získáno 27. 6 2012, z http://www.google.cz/books?id=YUc38_MCuhAC&dq=spectral+graph+theory+chung&lr=&hl=cs&source=gbs_navlinks_s

[7] Luxburg, U. v. (2007). *A Tutorial on Spectral Clustering.* Získáno 11. 6 2012, z Springer.com: http://www.springerlink.com/index/jq1g17785n783661.pdf

[8] M. Gu, H. Z. (2001). *Spectral Relaxation Models And Structure Analysis For K-Way Graph Clustering And Bi-Clustering.* Získáno 27. 6 2012, z CiteSeer: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.2657

[9] *OpenCV Wiki*. (nedatováno). Získáno 27. 6 2012, z Welcome, OpenCV Wiki: http://opencv.willowgarage.com/wiki/

[10] Scott White, P. S. (2005). A spectral clustering approach to finding communities in graph. *Proceedings of the Fifth SIAM International Conference on Data Mining* (stránky 274-285). California: Society for Industrial and Applied Mathematics.

[11] Stephen Guattery, G. L. (12 1994). *On the performance of spectral graph partitioning methods.* Získáno 17. 6 2012, z Carnegie Mellon University: http://www.cs.cmu.edu/afs/.cs.cmu.edu/Web/People/glmiller/Publications/Papers/GuMi94-tr.pdf